MICROCOPY RESOLUTION TEST CHART

LEVEL

ADA085160

DTIC
ELECTE
JUN 4 1980

C

A MAXIMAL FLOW APPROACH TO DYNAMIC ROUTING IN COMMUNICATION NETWORKS

Mario Jodorkovsky
Adrian Segall

FILE COPY

80 6 3 030

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A085160	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle) A Maximum Flow Approach to Dynamic Routing in Communication Networks	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER LIDS-R-988

7. AUTHOR(s) Mario Jodorkovsky Adrian Segall	8. CONTRACT OR GRANT NUMBER(s) Contract N00014-75-C-1183 N00014-77-C-0532

9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT Laboratory for Information and Decision Systems Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code No. 5T10 ONR Identifying No. 049-383

11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209	12. REPORT DATE May 1980
	13. NUMBER OF PAGES 114 pages

14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

   This work presents a new approach for building the feedback solution for the minimum delay dynamic message routing problem for single destination networks.
   The approach fully exploits the special structure of the constraint matrices obtained in the dynamic state space model suggested in previous works, by transforming every linear program arising from the necessary conditions, into a maximal weighted flow problem.

DD $_{1 JAN 73}^{FORM}$ 1473   EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Taking advantage of several properties concerning the networks correspond-
ing to the linear programming problems, all theorems regarding certain
simplifying characteristics of the feedback solution that apply in the case
of single-destination networks with all unity weightings in the cost
functional are reproved in a simplified and more straightforward manner.

A compact algorithm for the construction of the feedback solution is
presented, the algorithm being implementable on networks of reasonable size.

A method for obtaining all solutions of the linear programming problems
required by the algorithm, based on the application of linear programming
techniques in networks is provided. The method is implemented by a computer
program and several examples are run to test its applicability.

In addition, a deep geometrical insight to every step of the algorithm
is given by deriving the explicit set of inequalities defining the problem
constraint figure in the state-velocity space.

The complexity of the problem is also analyzed, being exponential in
the number of the network nodes, thus giving an idea of the maximal network
size for which a full feedback solution can be obtained under the available
computational resources.

Accession For		
NTIS GRA&I		☑
DDC TAB		☐
Unannounced		☐
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or special	
A		

INSTRUCTIONS FOR PREPARATION OF REPORT DOCUMENTATION PAGE

RESPONSIBILITY. The controlling DoD office will be responsible for completion of the Report Documentation Page, DD Form 1473, in all technical reports prepared by or for DoD organizations.

CLASSIFICATION. Since this Report Documentation Page, DD Form 1473, is used in preparing announcements, bibliographies, and data banks, it should be unclassified if possible. If a classification is required, identify the classified items on the page by the appropriate symbol.

COMPLETION GUIDE

General. Make Blocks 1, 4, 5, 6, 7, 11, 13, 15, and 16 agree with the corresponding information on the report cover. Leave Blocks 2 and 3 blank.

Block 1. Report Number. Enter the unique alphanumeric report number shown on the cover.

Block 2. Government Accession No. Leave Blank. This space is for use by the Defense Documentation Center.

Block 3. Recipient's Catalog Number. Leave blank. This space is for the use of the report recipient to assist in future retrieval of the document.

Block 4. Title and Subtitle. Enter the title in all capital letters exactly as it appears on the publication. Titles should be unclassified whenever possible. Write out the English equivalent for Greek letters and mathematical symbols in the title (see "Abstracting Scientific and Technical Reports of Defense-sponsored RDT/E," AD-667 000). If the report has a subtitle, this subtitle should follow the main title, be separated by a comma or semicolon if appropriate, and be initially capitalized. If a publication has a title in a foreign language, translate the title into English and follow the English translation with the title in the original language. Make every effort to simplify the title before publication.

Block 5. Type of Report and Period Covered. Indicate here whether report is interim, final, etc., and, if applicable, inclusive dates of period covered, such as the life of a contract covered in a final contractor report.

Block 6. Performing Organization Report Number. Only numbers other than the official report number shown in Block 1, such as series numbers for in-house reports or a contractor/grantee number assigned by him, will be placed in this space. If no such numbers are used, leave this space blank.

Block 7. Author(s). Include corresponding information from the report cover. Give the name(s) of the author(s) in conventional order (for example, John R. Doe or, if author prefers, J. Robert Doe). In addition, list the affiliation of an author if it differs from that of the performing organization.

Block 8. Contract or Grant Number(s). For a contractor or grantee report, enter the complete contract or grant number(s) under which the work reported was accomplished. Leave blank in in-house reports.

Block 9. Performing Organization Name and Address. For in-house reports enter the name and address, including office symbol, of the performing activity. For contractor or grantee reports enter the name and address of the contractor or grantee who prepared the report and identify the appropriate corporate division, school, laboratory, etc., of the author. List city, state, and ZIP Code.

Block 10. Program Element, Project, Task Area, and Work Unit Numbers. Enter here the number code from the applicable Department of Defense form, such as the DD Form 1498, "Research and Technology Work Unit Summary" or the DD Form 1634, "Research and Development Planning Summary," which identifies the program element, project, task area, and work unit or equivalent under which the work was authorized.

Block 11. Controlling Office Name and Address. Enter the full, official name and address, including office symbol, of the controlling office. (Equates to funding/sponsoring agency. For definition see DoD Directive 5200.20, "Distribution Statements on Technical Documents.")

Block 12. Report Date. Enter here the day, month, and year or month and year as shown on the cover.

Block 13. Number of Pages. Enter the total number of pages.

Block 14. Monitoring Agency Name and Address (if different from Controlling Office). For use when the controlling or funding office does not directly administer a project, contract, or grant, but delegates the administrative responsibility to another organization.

Blocks 15 & 15a. Security Classification of the Report: Declassification/Downgrading Schedule of the Report. Enter in 15 the highest classification of the report. If appropriate, enter in 15a the declassification/downgrading schedule of the report, using the abbreviations for declassification/downgrading schedules listed in paragraph 4-207 of DoD 5200.1-R.

Block 16. Distribution Statement of the Report. Insert here the applicable distribution statement of the report from DoD Directive 5200.20, "Distribution Statements on Technical Documents."

Block 17. Distribution Statement (of the abstract entered in Block 20, if different from the distribution statement of the report). Insert here the applicable distribution statement of the abstract from DoD Directive 5200.20, "Distribution Statements on Technical Documents."

Block 18. Supplementary Notes. Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of (or by) . . . Presented at conference of . . . To be published in . . .

Block 19. Key Words. Select terms or short phrases that identify the principal subjects covered in the report, and are sufficiently specific and precise to be used as index entries for cataloging, conforming to standard terminology. The DoD "Thesaurus of Engineering and Scientific Terms" (TEST), AD-672 000, can be helpful.

Block 20. Abstract. The abstract should be a brief (not to exceed 200 words) factual summary of the most significant information contained in the report. If possible, the abstract of a classified report should be unclassified and the abstract to an unclassified report should consist of publicly- releasable information. If the report contains a significant bibliography or literature survey, mention it here. For information on preparing abstracts see "Abstracting Scientific and Technical Reports of Defense-Sponsored RDT&E," AD-667 000.

A MAXIMAL FLOW APPROACH

TO DYNAMIC ROUTING IN COMMUNICATION NETWORKS

Mario Jodorkovsky and Adrian Segall

Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa, Israel

Contents

Abstract

This work presents a new approach for building the feedback solution for the minimum delay dynamic message routing problem for single destination networks.

The approach fully exploits the special structure of the constraint matrices obtained in the dynamic state space model suggested in previous works, by transforming every linear program arising from the necessary conditions, into a maximal weighted flow problem.

Taking advantage of several properties concerning the networks corresponding to the linear programming problems, all theorems regarding certain simplifying characteristics of the feedback solution that apply in the case of single-destination networks with all unity weightings in the cost functional are reproved in a simplified and more straightforward manner.

A compact algorithm for the construction of the feedback solution is presented, the algorithm being implementable on networks of reasonable size.

A method for obtaining all solutions of the linear programming problems required by the algorithm, based on the application of linear programming techniques in networks is provided. The method is implemented by a computer program and several examples are run to test its applicability.

In addition, a deep geometrical insight to every step of the algorithm is given by deriving the explicit set of inequalities defining the problem constraint figure in the state-velocity space.

The complexity of the problem is also analyzed, being exponential in the number of the network nodes, thus giving an idea of the maximal network size for which a full feedback solution can be obtained under the available computational resources.

Glossary of Notations

Notation		Definition
N	-	set of network nodes not including the destination node
L	-	set of network links
(i,k)	-	directed link from node i to node k
c_{ik}	-	capacity of link (i,k)
$\underline{x}$	-	vector of state variables
$\underline{y}$	-	vector of velocity of state variables $(-\underline{\dot{x}})$
$\underline{u}$	-	vector of control variables
$\underline{\lambda}$	-	vector of costates
U	-	constraint figure in $\underline{u}$-space
Y	-	constraint figure in $\underline{y}$-space
I_p	-	set of states travelling on interior arcs on $[t_p, t_{p+1})$
B_p	-	set of states travelling on boundary arcs on $[t_p, t_{p+1})$
L_p	-	set of states leaving the boundary at t_p backwards in time
R_p	-	feedback control region constructed from optimal trajectories on the segment $[t_p, t_{p+1})$
Y_p	-	set of operating points on $[t_p, t_{p+1})$
$(X, \bar{X})$	-	a minimal cut of the network
b_i	-	demand for flow in node i
b_s	-	supply of flow to the network
σ_p	-	cardinality of I_p
ρ_p	-	cardinality of L_p
$H(\tau)$	-	the Hamiltonian at τ
$C_o(\cdot)$	-	Convex Hull
$E(i)$	-	Collection of nodes k such that $(i,k) \in L$
$I(i)$	-	Collection of nodes ℓ such that $(\ell,i) \in L$

Section 1

INTRODUCTION

In [S1] a state space model for dynamic routing in data communication networks is suggested. The main feature of this model is that it permits to express the delay experienced by the messages travelling in the network in terms of state and control variables describing the problem instead of models based on queueing theory. The latter requires explicit closed-form expressions for the average delays which can be found analytically only for very special distributions and dependence relationships. The model also permits to develop closed loop strategies for the message routing problem and can handle transients by changing the routing policy in a dynamic fashion.

We begin by presenting a brief description of the model, simplified to the case of single destination networks with zero inputs. Consider first the following notations:

$N = \{1,2,\ldots,n\}$ is the set of network nodes (not including the destination node).

d = destination node.

$L = \{(i,k)/i,k \in N \cup d$ and there is a direct link connecting i to $k\}$ is the set of network links.

$E(i)$ = collection of nodes k such that $(i,k) \in L$.

$I(i)$ = collection of nodes ℓ such that $(\ell,i) \in L$.

All links of the network are taken to be unidirectional. Now, looking at the network from a macroscopic point of view, the number of messages in each node can be approximated by a continuous variable called "amount of traffic". The state variables of the system are defined as follows:

$x_i(t) \triangleq$ amount of traffic at node i, at time t, where $i \in N$.

The control variables are defined as:

$u_{ik}(t) \triangleq$ amount of traffic on link (i,k) at time t where $(i,k) \in L$.

The dynamics of the system are given by the equations describing the rate of change of the contents of each node, namely:

$$\dot{x}_i(t) = - \sum_{k \in E(i)} u_{ik}(t) + \sum_{\ell \in I(i)} u_{\ell i}(t) \qquad (1.1)$$

$$\forall \; i \in N \; .$$

The constraints are:

$$x_i(t) \geq 0 \; , \qquad (1.2a)$$

and

$$U: \; 0 \leq u_{in}(t) \leq c_{ik} \; , \qquad (1.2b)$$

where

$c_{ik} \triangleq$ capacity of link (i,k) in units of traffic/unit time, where $(i,k) \in L$.

The cost functional is taken to be the total delay experienced by the messages travelling in the network, starting at a given time t_o and ending at a time t_f when the network is emptied, i e $x_i(t_f) = 0, \; \forall \; i \in N$. The expression for the above quantity is

$$J = \int_{t_o}^{t_f} [\sum_{i \in N} x_i(t)]dt \; . \qquad (1.3)$$

From now on, the various variables of the model will be represented in vector form as follows: Denoting by $\underline{x}$ and $\underline{u}$ the respective concatenations of state variables and controls, to every differential constraint in (1.1) corresponds a vector $\underline{b}_i$ such that:

$$\dot{x}_i = \underline{b}_i \underline{u} \quad ,$$

where the vectors $\underline{b}_i$ are the rows of the incidence matrix B of the network.

Now we express the linear optimal control problem with linear state and control variable inequality constraints representing the data communication network closed loop dynamic routing problem as stated in [S1]:

Find the set of controls $\underline{u}$ as a function of time and state,

$$\underline{u}(t) \triangleq \underline{u}(t,\underline{x}) \quad , \qquad t \varepsilon [t_o, t_f] \quad ,$$

that brings a given initial condition $\underline{x}(t_o) = \underline{x}_o$ to the final condition $\underline{x}(t_f) = 0$ and minimizes the cost functional (1.3) subject to the dynamics (1.1) and to the state and control variable inequality constraints (1.2).

It is well know that in most optimal control problems it is quite difficult to obtain feedback solutions. In our case however, owing to the linearity of the problem, we can cover the entire state space with optimal controls by solving just a comprehensive set of linear programs. In [S1] an approach is suggested, by way of a simple example for constructing the feedback solution. In [M1], [M2] and [M3] this approach is elaborated upon by developing the so called Constructive Dynamic Programming Algorithm for the construction of the feedback solution. We proceed now with a brief presentation of the results obtained in the above works and the conceptual lines of the Constructive Dynamic Programming Algorithm in order to provide the reader with basic notions for the understanding of the present report. We begin by presenting the necessary and sufficient conditions of optimality:

Theorem 1.1: (See [M1, pp. 53-63] or [M2, pp. 13-20]).

Let the scalar functional h be defined as follows:

$$h(\underline{u}(t),\underline{\lambda}(t)) \triangleq \underline{\lambda}^T(t)\dot{\underline{x}}(t) = \underline{\lambda}^T(t)B\underline{u}(t) \quad .$$

A necessary and sufficient condition for the control law $\underline{u}^*(\cdot) \in \mathcal{U}$ to be optimal for problem (1.1) - (1.3) is that it minimizes h pointwise in time, namely:

$$\underline{\lambda}^T(t)B\underline{u}^*(t) \leq \underline{\lambda}^T(t)B\underline{u}(t) \tag{1.4}$$

$$\forall \underline{u}(t) \in \mathcal{U}, \quad \forall t \in [t_o, t_f] \quad .$$

The costate $\underline{\lambda}(t)$ is possibly a discontinuous function which satisfies the following differential equations:

$$-d\lambda_i(t) = dt + d\mu_i(t), \quad \forall i \in N \quad t \in [t_o, t_f] , \tag{1.5}$$

where componentwise $d\mu_i(t)$ satisfies the complementary slackness conditions:

$$\left. \begin{array}{r} x_i(t)d\mu_i(t) = 0 \\ d\mu_i(t) \leq 0 \end{array} \right\} \begin{array}{l} \forall t \in [t_o, t_f] \\ \forall i \in N \end{array} \quad . \tag{1.6}$$

The terminal boundary condition for the costate differential equation is

$$\underline{\lambda}(t_f) = \underline{0} \quad \text{free} \quad , \tag{1.7}$$

and the transversality condition is

$$\underline{\lambda}^T(t_f)\underline{\dot{x}}(t_f) = 0 \quad . \tag{1.8}$$

Finally, the function h is everywhere continuous, i.e.:

$$h(\underline{u}(t^-), \underline{\lambda}(t^-)) = h(\underline{u}(t^+), \underline{\lambda}(t^+)), \quad \forall t \in [t_o, t_f] \quad .$$

$\square$ Theorem 1.1

From inequality (1.4) of the necessary and sufficient conditions we see that the optimal control function $\underline{u}^*(\cdot)$ is given at every time $\tau \in [t_o, t_f]$ by the solution of the linear program

$$\underline{u}^*(\tau) = \operatorname*{ARGMIN}_{\underline{u}(\tau) \in \mathcal{U}} [\underline{\lambda}^T(\tau)B\underline{u}(\tau)] \quad . \tag{1.9}$$

From (1.9), and owing to the structure of the incidence matrix B, it follows that there always exists an optimal solution for which the controls are piecewise constant in time. Moreover, since the equations governing the dynamics of the system are linear, the corresponding state trajectories have piecewise constant slopes. From the nature of the controls follows that every optimal trajectory may be characterized by a finite number of parameters. These parameters are:

$$U(\underline{x}) \triangleq \{\underline{u}_o, \underline{u}_1, \ldots, \underline{u}_{f-1}\}$$

and

$$T(\underline{x}) \triangleq \{t_o, t_1, \ldots, t_f\} \quad,$$

where $U(\underline{x})$ is the sequence of optimal controls, $T(\underline{x})$ is its associated control switch time sequence, and the element $\underline{u}_p$ is the optimal control on $t \in [t_p, t_{p+1}]$, $p \in [0, 1, \ldots, f-1]$.

Moreover, every segment of an optimal trajectory, say the segment $[t_p, t_{p+1}]$ is characterized by the following parameters:

$$B_p = \{x_i / x_i(\tau) = 0 \quad \forall \tau \in [t_p, t_{p+1})\}$$

and

$$I_p \triangleq \{x_i / x_i(\tau) > 0 \quad \forall \tau \in [t_p, t_{p+1})\} \quad,$$

that is, B_p and I_p are the sets of states travelling on boundary arcs and interior arcs respectively on $[t_p, t_{p+1})$. Now, the main fact following from the nature of the optimal trajectories is that the state space can be divided into regions, each one being a convex polyhedral cone, when to every point of a specific region corresponds an identical set of optimal controls. This is exactly the reason that makes the construction of a feedback solution possible. The above regions are referred to as "feedback control regions" and are denoted by R.

The problem is then to construct these feedback control regions, thereby specifying the optimal control for every point of the state space. Now,

considering the special geometric characterization of the feedback space, and thinking in the spirit of dynamic programming, let us look at the optimal trajectories backwards in time, beginning at t_f. We will then see a sequence of states leaving the boundary (perhaps two or more at a time) and varying with constant slopes. Now, in [M2] it is proven that if any state variable, say x_i, is strictly positive on the last time interval $[t_{f-1}, t_f]$ of an optimal trajectory, then $\lambda_i(t_f) = 0$. Moreover, by (1.6) we know that for this state we have $d\mu_i(\tau) = 0 \ \forall \tau \in [t_{f-1}, t_f]$ so that by (1.5) we obtain $\lambda_i(\tau) = -1 \ \forall \tau \in [t_{f-1}, t_f]$.

Now suppose we had decided to check if there exist optimal trajectories in the state space with a specified set of states I_{f-1} travelling on interior arcs in the last interval and with the remaining states travelling on boundary arcs. Since we know the costates corresponding to the states in I_{f-1} (by the above arguments) and using (1.9), these trajectories may be found by solving the following linear program:

Find all:

$$
\left.
\begin{aligned}
& \underline{u}^* = \operatorname*{ARGMIN}_{\underline{u} \in U} \sum_{x_i \in I_{f-1}} \lambda_i(\tau)\dot{x}_i = \operatorname*{ARGMIN}_{\underline{u} \in U} \sum_{x_i \in I_{f-1}} \lambda_i(\tau)\underline{b}_i\underline{u} \\[2mm]
\text{s.t.} \quad & \dot{x}_i \leq 0 \quad \forall x_i \in I_{f-1} \\[2mm]
& \dot{x}_j = 0 \quad \forall x_i \in B_{f-1} \\[2mm]
& \tau \in (t_f, t_{f-1})
\end{aligned}
\right\} \quad (1.10)
$$

The linear program (1.10) is called the constrained optimization problem and its solutions are trajectories only provided that there exist values $\lambda_j \ \forall x_j \in B_{f-1}$ that satisfy the necessary conditions (1.5) - (1.6). Moreover, the solutions of (1.10) provide optimal directions $\dot{x}_i^* \ \forall x_i \in I_{f-1}$ in the state space, defining the convex polyhedral cone corresponding to the feedback

control region characterized by the set of controls that solves (1.10). Now, solving such a linear program for every possible combination of state variables leaving the boundary (backwards in time) at t_f, we obtain feedback control regions containing all points from which the origin can be reached while maintaining sets I_{f-1} away from the boundary for all $t < t_f$. In fact the linear program (1.10) must be solved parametrically in time until the control changes. A change in the control defines a hyperplane passing through the origin called "breakwall". By solving (1.10) until the control does not break any more, we obtain a convex polyhedral cone divided by breakwalls into a finite number of regions. Each region is characterized by a specific control set and is referred to as "break feedback control region". The last region constructed when solving (1.10), (i.e. when there are no more breaks in the control) is called "non-break feedback control region".

In order to continue with the description of the algorithm, let us introduce the following definition:

$$L_p \triangleq \{x_i/x_i \in B_p \text{ and } x_i \text{ is designated to leave the boundary}$$
$$\text{backward in time at } t_p\} \ .$$

Now, in order to cover the whole state space with optimal controls, we must take every feedback control region already constructed and allow every possible combination of states still travelling on the boundary to leave it, backward in time. In general, this step is performed as follows: Denoting by R_p the feedback control region constructed from the optimal trajectories on the segment $[t_p, t_{p-1})$, pick a set of states of B_p, say L_p and partition R_p into "subregions" with respect to L_p. The concept "subregion" will be explained later. In order to let L_p leave the boundary at a certain time t_p (called boundary junction time), we must find the costate values $\lambda_i(t_p) \ \forall x_i \in L_p$ that allow the departure of the states in L_p from the

boundary while still being optimal. The above costate values are referred to as "leave-the-boundary costate values". In geometrical terms the above co-state values are found as follows:

By defining

$$y \triangleq -\dot{x} = -B\underline{u}$$

and

$$\mathcal{Y} \triangleq \{\underline{y} \in \mathbb{R}^n / \underline{u} \in \mathcal{U}\} \quad,$$

we can transform the linear program (1.9) into the following linear program with decision vector $\underline{y}(\tau)$:

$$\underline{y}^*(\tau) = \underset{y(\tau) \in \mathcal{Y}}{\text{ARGMAX}} \; \underline{\lambda}^T(\tau)\underline{y}(\tau) \quad, \tag{1.11}$$

so that at t_p we can consider the hyperplane given by $z(t_p) = \underline{\lambda}^T(t_p)\underline{y}$ (called the Hamiltonian) tangent to $\mathcal{Y}$ (called the $\underline{y}$-constraint figure) that provides points of tangency Y_p (called operational points) that are in fact the optimal directions in the state space defining the feedback control region R_p. Now in order to find the leave-the-boundary costates for L_p, we rotate the Hamiltonian around Y_p until we touch a surface of tangency of $\mathcal{Y}$ called L_p-positive face, having at least one point with

$$y_i > 0 \quad \forall x_i \in L_p \quad.$$

The new orientation of the Hamiltonian gives the desired leave-the-boundary costates and now we solve a linear program of the form:

Find all:

$$\left. \begin{array}{l} \underline{u}^* = \underset{\underline{u} \in \mathcal{U}}{\text{ARGMIN}} \; \sum_{x_i \in I_{p-1}} \lambda_i(\tau)\dot{x}_i = \text{ARGMIN} \; \sum_{x_i \in I_{p-1}} \lambda_i(\tau)\underline{b}_i\underline{u} \\[2ex] \text{s.t.} \\ \dot{x}_i \leq 0 \quad \forall x_i \in I_{p-1} = I_p \mathcal{U} L_p \\[1ex] \dot{x}_i = 0 \quad \forall x_i \in B_{p-1} = B_p / L_p \\[1ex] \qquad \forall \tau \in (t_p, t_{p-1}) \quad, \end{array} \right\} \tag{1.12}$$

where (1.12) is again solved parametrically in time until the control breaks.

A subregion of R_p is the set of all points which, when taken as the point of departure of L_p, result in a common set of optimal controls. Clearly, such a partitioning of R_p may exist since to two distinct points in R_p correspond different leave-the-boundary costates. In geometrical terms, we could see the Hamiltonian rotating according to the change in the costates when the states are travelling in R_p. When allowing L_p to leave the boundary from distinct points of R_p, the Hamiltonian could meet different L_p-positive faces, therefore giving different leave-the-boundary costate values.

The steps described for the construction of feedback control regions form the so-called Constructive Dynamic Programming Algorithm as stated in [M1], [M2] and [M3]. Several features make the algorithm to be a conceptual one rather than an implementable one. Among them, breaks in the optimal controls, the existence of the subregions just pointed out, non-uniqueness of the leave-the-boundary costates, and non-global optimality of certain sequences, in addition to computational complexities associated with the algorithm, for instance the problem of finding all solutions to linear programs. However, it turns out that when dealing with single destination networks, and when no priorities are assigned to the nodes (that is the cost functional has unity weightings), many simplifications are attained, leading to a compact and implementable algorithm at least for moderate size networks.

In the present paper, we deal with the construction of the feedback solution to the minimum delay dynamic message routing problem for single destination networks, with zero inputs and when the cost functional has unity weightings. The approach used to tackle the problem is based on the trans-

formation of every linear program into a maximal weighted flow problem. The
transformation permits not only to develop a compact algorithm for obtaining
the feedback solution, but also to prove in a straightforward manner all the
simplifying features characterizing the feedback space, to develop a suitable
method for finding all the solutions to the linear programming problems and
to obtain explicitly the constraint figure in the state velocity space for
the geometrical understanding of the algorithm. The approach also permits
to gain insight into the complexity of the problem, thus providing the
essential information needed to estimate the maximal size of the networks for
which a complete feedback solution can be obtained under the available compu-
tational resources.

One of the most interesting features of the algorithm presented here
is that although the efficiency of the method for obtaining all optimal solu-
tions to the linear programs is reduced by the high degeneracy of the problems
at hand, this is compensated by the fact that the higher the degree of
degeneracy, the lower is the number of linear programs that have to be solved.
Moreover, the algorithm requires to check all possible sequences of states
leaving the boundary backward in time or, in other words, all the possible
trajectories in the state space to insure the complete covering of the state
space. Again the higher the degree of degeneracy, the lower the number of
such sequences that have to be actually checked, thus obtaining a further
reduction in the complexity of the algorithm

The organization of the paper is as follows:

In Section 2, a general description of the algorithm is presented and
the transformation of the linear programs into maximal weighted flow problems
is introduced. Section 3 is devoted to the theoretical results obtained,
namely, the structure of the y-constraint figure and the proofs regarding all

simplifying features of the feedback solution that apply in the case of single destination networks.

In Section 4 the algorithm is presented in a compact form, the method for obtaining all solutions to the linear programs is described and an example of the construction of the feedback solution is given.

Finally, in Section 5 a brief discussion about the contributions of this work is carried out and topics for further research in the area are suggested.

Section 2

GENERAL DESCRIPTION OF THE ALGORITHM

We begin with a brief preview of this section. In part A we present a simplified mathematical description of the Constructive Dynamic Programming Algorithm for building the feedback solution for the minimum delay dynamic message routing problem for single destination networks with all unity weightings in the cost functional. The algorithm consists of two steps. In the first step, we construct feedback control regions resulting from the states leaving the boundary at the final time (backward in time). The method to construct these regions is depicted in parts B and C of this section. In the second step, the rest of the state space is filled with optimal controls by constructing the feedback control regions resulting from states leaving the boundary, starting from already constructed regions. Part D of this section deals with the construction of the above regions.

A. **Mathematical Statement of the Algorithm**

In [M2], the following properties associated with the Constructive Dynamic Programming Algorithm are mentioned:

(a) Non-global optimality of certain sequences of state variables leaving the boundary backwards in time.

(b) Non-uniqueness of leave-the-boundary costates.

(c) Subregions.

(d) Return of state variables to boundary backwards in time.

(e) Breaks in the optimal control between boundary junction times.

In [M3], a discussion of the above properties is presented in geometrical terms. These problems complicate the formulation of a computational scheme to implement the algorithm. However, as we will prove in Section 3 of this paper, it turns out that these problems do not apply in the case of single destination networks with all unity weightings in the cost functional, thus permitting the following simplified statement of the algorithm:

Step 1: Solve the following series of linear programming problems:

Find all

$$u^* = \underset{\underline{u} \in U}{ARGMIN} \sum_{x_i \in L_f} \dot{x}_i = \underset{\underline{u} \in U}{ARGMIN} \sum_{x_i \in L_f} b_i \underline{u}$$

s.t.

$$\dot{x}_i \le 0 \quad \forall\, x_i \in L_f$$

$$\dot{x}_j = 0 \quad \forall\, x_j \in B_{f-1}$$

(2.1)

for all $\qquad L_f \subset \{x_i / i \in N\}$

when N denotes the set of nodes not including the destination.

Step 2: For all feedback control regions R_p:

(a) Calculate the leave-the-boundary costate values $\lambda_i(t_p)$ for all $x_i \in B_p$, t_p being an arbitrary boundary junction time.

(b) Solve the following series of linear programming problems:

Find all:

$$u^* = \underset{\underline{u} \in U}{ARGMIN} \sum_{x_i \in I_{p-1}} \lambda_i(t_p)\dot{x}_i = \underset{\underline{u} \in U}{ARGMIN} \sum_{x_i \in I_{p-1}} \lambda_i(t_p)b_i\underline{u}$$

s.t.

$$\dot{x}_i \le 0 \quad \forall\, x_i \in I_{p-1}$$

$$\dot{x}_j = 0 \quad \forall\, x_j \in B_{p-1}$$

(2.2)

for all $\qquad L_p \subset B_p$.

As we said before, in the first step, feedback control regions resulting from the states leaving the boundary at t_f, are constructed. Note that in problem (2.1), the cost function has unity weightings. The reason is that by Corollary 2 in [M2], $\lambda_i(t_f) = 0 \; \forall x_i \in L_f$, and by the necessary and sufficient conditions for optimality (see [M2, pp. 13-17]), $\dot{\lambda}_i(\tau) = -1 \; \forall x_i \in L_f$. Moreover, since there are no breaks between boundary junction times, the costates corresponding to states in L_f are equal for all $\tau \in (-\infty, t_f)$ so that at every time in this interval the cost function is the same after normalization. From this fact follows that at every time in $(-\infty, t_f)$ we obtain the same set of optimal solutions to problem (2.1), so that the time plays no role here. Notice also, that since each set of state variables leaving the boundary corresponds to a globally optimal solution, every solution to (2.1) gives an optimal trajectory.

In the second step, feedback control regions resulting from states leaving the boundary from previously constructed regions are built. Since there is only one subregion per region, it follows that from every point of a given feedback control region, states leave the boundary with the same control set and therefore boundary junction times may be chosen arbitrarily. The above reasoning together with the fact that the leave-the-boundary costates are unique for a given boundary junction time, and with the fact that no state variable is ever required by optimality to increase forward in time, justify the simplifications obtained in the statement of the algorithm.

The first step is carried out as follows: First, the linear program corresponding to the leave-the-boundary of all states is solved. Second, the linear programs corresponding to (n-1) states leaving the boundary are solved (when n is the number of nodes in the network not including the destination node), and so on. As it will be shown later, this way of solving the first

step enables in general a considerable reduction in the number of linear programs that have to be solved. Moreover, it will also be shown later that in fact the second step does not require solving any linear programs, thus greatly reducing the complexity of the algorithm.

B. All States Leave the Boundary

According to (2.1), the linear programming problem we have to solve for this case is:

Find all:

$$\underline{u}^* = \underset{\underline{u} \in U}{\text{ARGMIN}} \sum_{i \in N} \dot{x}_i = \underset{\underline{u} \in U}{\text{ARGMIN}} \sum_{i \in N} \underline{b}_i \underline{u} \quad , \qquad (2.3)$$

s.t.

$$\dot{x}_i \leq 0 \quad \forall i \in N \quad , \qquad (2.3a)$$

$$U = \begin{cases} 0 \leq u_{jk} \leq c_{jk} \\ \\ \forall (j,k) \in L \end{cases} \qquad (2.3b)$$

Adding slack variables $y_i \geq 0$ to the differential constraints (2.3a), problem (2.3) is transformed into:

Find all:

$$\underline{u}^* = \underset{\underline{u} \in U}{\text{ARGMIN}} \sum_{i \in N} \underline{b}_i \underline{u} = \underset{\underline{u} \in U}{\text{ARGMAX}} \sum_{i \in N} y_i \quad , \qquad (2.4)$$

s.t.

$$\begin{aligned} \dot{x}_i + y_i &= 0 \\ & \qquad\qquad \forall i \in N \quad , \\ y_i &\geq 0 \end{aligned} \qquad (2.4a)$$

$$U = \begin{cases} 0 \leq u_{jk} \leq c_{jk} \\ \\ \forall (j,k) \in L \end{cases} \qquad (2.4b)$$

Notice that in terms of the variable "y", the minimization problem (2.3) is

transformed into the maximization problem (2.4).

Suppose we are interested in finding only one solution to problem (2.4). In order to do this, we transform the linear program (2.4) into the following Maximal Flow Problem:

$$\text{Maximize} \quad F = \sum_{i \in N} y_i \ , \tag{2.5}$$

s.t.

$$- \sum_{j \in E(i)} u_{ij} + \sum_{k \in I(i)} u_{ki} + y_i = 0$$

$$\forall \ i \in N \tag{2.5a}$$

$$y_i \geq 0$$

$$\underline{u} \in U = \begin{cases} 0 \leq u_{jk} \leq c_{jk} \\ \\ \forall \ (j,k) \in L \end{cases} . \tag{2.5b}$$

The network corresponding to problem (2.5) is formed by adding to the original network a new node called the "source", and n links with no capacity constraints connecting the source with each node of the network (except for the destination node). The slack variable "y_i" represents the flow on the link connecting the source with node i. F represents the total flow into (and out of) the network (see Fig. 2.1).
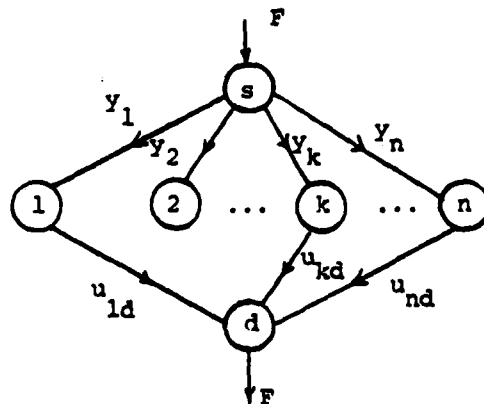
s = source node
d = destination node

The links connecting
the different nodes
are omitted here to
avoid causing havoc
in the figure.

Figure 2.1 - Network representing Problem (2.5)

The transformation of the linear program into a Maximal Flow Problem is the fundamental step required for the proofs of the theorems regarding the simplifying properties of the feedback solution. In Appendix A, we give basic concepts of graph theory that we use in the development of the algorithm and in the proofs of the theorems.

Now we continue with the solutions of problem (2.5). Our aim is to find one solution to problem (2.5) by means of the Maximal Flow Algorithm and then, without changing the flow achieved in the first optimal solution, to perform all possible pivots on the network in order to find all the remaining optimal solutions of problem (2.5). Notice that since we are seeking extremal optimal solutions, we require that the first one will be an "extended basic optimal solution", (see Appendix A). In this way every solution obtained by pivoting will also be an extended basic optimal solution, thus representing an extremal point in the optimization space. It turns out that finding a first "extended basic optimal solution" to problem (2.5) is trivial. From the fact that the links corresponding to the "y" variables do not have capacity constraints follows that there exists only one minimal cut in the network (denoted by $(X,\bar{X})$), and it is composed by all the links connecting the nodes with the destination; thus:

$$X = \{s,1,2,\ldots,n\} \quad ,$$
$$\bar{X} = \{d\} \ .$$

Then we choose as the first optimal solution the following flows:

$$y_i = \begin{cases} c_{id} & i \in I(d) \\ \\ 0 & \text{otherwise} \end{cases}$$

$$\text{(2.6)}$$

$$u_{id} = c_{id} \quad \forall \, i \in I(d)$$
$$u_{ij} = u_{ki} = 0 \quad j \in E(i), \, k \in I(i)$$
$$\forall \, i \in N$$

and

$$F = \sum_{i \varepsilon I(d)} c_{id} \quad . \qquad (2.6a)$$

Clearly, (2.6) is an "extended basic solution" if the variables y_i are declared basic and the remaining variables are declared non-basic. Notice that if there is no direct connection between a node, say node i, and the destination node, then the variable y_i is basic with value zero. Now, since networks have generally many nodes not connected directly with the destination we are dealing here with linear programming problems of a high degree of degeneracy. The degeneracy problem lowers the efficiency of linear programming methods, however in our case, the higher the degree of degeneracy, the lower is the number of linear programs we have to solve. This subject will be discussed in Part C of this section.

Recall that after finding the first "extended optimal basic solution" we are interested in maintaining the achieved flow to assure that every solution obtained by pivoting operations on the network will be optimal. But maintaining maximal flow in the network implies that at every solution the flow on the minimal cuts will be constant and maximal; thus, before pivoting we can reduce the number of variables of the problem by transforming the network to the one depicted in Figure 2.2:
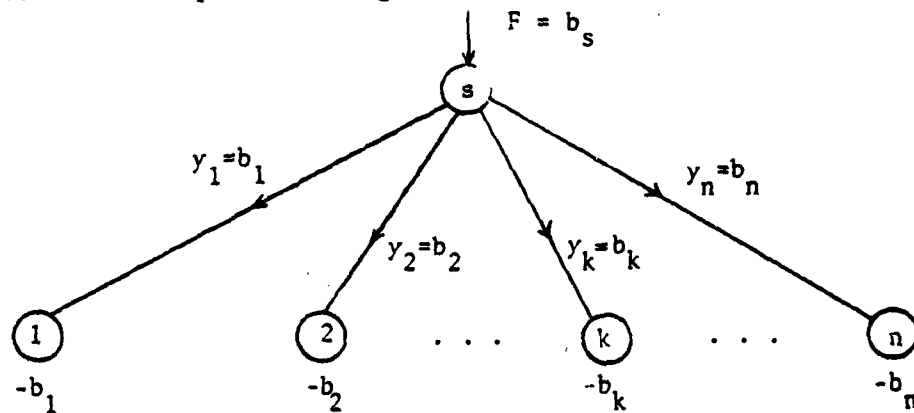
Figure 2.2 - Network configuration before the pivoting operation.

In Figure 2.2, b_i accounts for the "demand" of flow in node i, and b_s is the "supply" of flow to the network where:

$$b_i = \begin{cases} c_{id} & i \in I(d) \\ & \hspace{1cm} \forall i \in N \\ 0 & \text{otherwise} \end{cases}$$

$$b_s = \sum_{i \in I(d)} c_{id} \; .$$

(2.7)

Clearly, the reduction in the number of links facilitates the work of finding all optimal solutions. After the reduction, the remaining optimal solutions are found by means of the algorithm we present in Section 4.

C. A Subset of States Leaves the Boundary

In this case, we let a subset $L_f \subset \{x_i / i \in N\}$, such that $L_f \neq \{x_i / i \in N\}$ to leave the boundary at t_f, while all other states B_{f-1} remain on the boundary. By (2.1), the linear program to be solved is:

Find all:

$$\underline{u}^* = \underset{\underline{u} \in U}{\text{ARGMIN}} \sum_{x_i \in L_f} \dot{x}_i = \underset{\underline{u} \in U}{\text{ARGMIN}} \sum_{x_i \in L_f} b_i \underline{u} \qquad (2.8)$$

s.t.

$$\dot{x}_i \leq 0 \quad \forall \; x_i \in L_f \quad ,$$

$$\dot{x}_j = 0 \quad \forall \; x_j \in B_{f-1} \quad .$$

(2.8a)

Transforming (2.8) into a Maximal Flow Problem, we have:

$$\text{Maximize} \quad F = \sum_{x_i \in L_f} y_i \qquad (2.9)$$

s.t.

$$- \sum_{j\epsilon E(i)} u_{ij} + \sum_{k\epsilon I(i)} u_{ki} + y_i = 0 \quad \forall x_i \epsilon L_f$$

$$\sum_{j\epsilon E(i)} u_{ij} - \sum_{k\epsilon I(i)} u_{ki} = 0 \quad \forall x_i \epsilon B_{f-1}$$

$$y_i \geq 0 \quad \forall x_i \epsilon L_f$$

$$\underline{u} \epsilon \, U = \begin{cases} 0 \leq u_j k \leq c_{jk} \\ \\ \forall \, (j,k) \epsilon L \end{cases}$$

(2.9a)

The network corresponding to problem (2.9) is formed by adding a new node called the "source" to the original network, as in Part B of this section. The difference here is that we add links without capacity constraints connecting the source only with those nodes that correspond to state variables of L_f. Nodes corresponding to state variables in B_{f-1} will be not connected with the source. For example, Figure 2.3 shows a network with five nodes corresponding to the case of two states leaving the boundary:
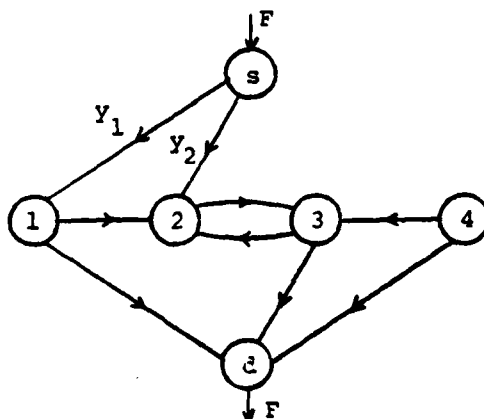
Figure 2.3 - Example of network when only some states leave the boundary.

Our aim is, as in Part A, to find a first "extended basic optimal solution" to problem (2.9) and then by pivoting to find the remaining solutions. But here we cannot obtain the first optimal solution by inspection since it is

not trivial to localize a minimal cut in the network. Moreover, we can obtain
an optimal solution by applying the Maximal Flow Algorithm to the network, but
this solution may not necessarily be an extended basic one. In order to over-
come this difficulty we present now two theorems and a corollary that will
assist us in developing a method to solve the problem when only some states
leave the boundary. In addition these results will show that in general there
are many combinations of state variables leaving the boundary for which it is
not necessary to solve a linear programming problem.

Theorem 2.1

Consider the following two linear programming (or Maximal Flow) problems:

Problem I

$$\text{Maximize} \quad F_1 = \sum_{x_i \epsilon L_f'} y_i$$

s.t.

$$-\sum_{j \epsilon E(i)} u_{ij} + \sum_{k \epsilon I(i)} u_{ki} + y_i = 0 \quad \forall x_i \epsilon L_f'$$

$$\sum_{j \epsilon E(i)} u_{ij} - \sum_{k \epsilon I(i)} u_{ki} = 0 \quad \forall x_i \epsilon B_{f-1}'$$

$$y_i \geq 0 \quad \forall x_i \epsilon L_f'$$

$$\underline{u} \epsilon U \quad .$$

Problem II

$$\text{Maximize} \quad F_2 = \sum_{x_i \epsilon L_f} y_i$$

$$-\sum_{j \epsilon E(i)} u_{ij} + \sum_{k \epsilon I(i)} u_{ki} + y_i = 0 \quad \forall x_i \epsilon L_f$$

$$\sum_{j \epsilon E(i)} u_{ij} - \sum_{k \epsilon I(i)} u_{ki} = 0 \quad \forall x_i \epsilon B_{f-1}$$

$$y_i \geq 0 \quad \forall x_i \epsilon L_f$$

$$\underline{u} \epsilon U \quad ,$$

where $L_f \subset L_f' \subset \{x_i / i \epsilon N\}$.

If in problem I exists a basic optimal solution with $\{y_j = 0,$ $\forall x_j \in L'_f/L_f\}$, then all the basic optimal solutions of II are basic optimal solutions of I. Moreover, for the networks representing the above problems, all minimal cuts of II are minimal cuts of I.

Proof

Take the solution of I with $\{y_j = 0,\ \forall x_j \in L'_f/L_f\}$. This solution satisfies the constraints of II and clearly $\underset{u \in U}{\text{Max}}\ F_1 = \underset{u \in U}{\text{Max}}\ F_2$. Now take all basic optimal solutions of II and add new variables $\{y_i \geq 0,\ \forall x_i \in L'_f/L_f\}$ as non-basic ones with value zero. Clearly all these solutions satisfy the constraints of I. Moreover, since $\text{Max}\ F_1 = \text{Max}\ F_2$, and the new variables were added as non-basic ones, the solutions are also basic optimal solutions of I.

For the corresponding networks, take now all solutions of I with $\{y_j = 0,\ \forall x_j \in L'_f/L_f\}$. Removing the links corresponding to $\{y_j/x_j \in L'_f/L_f\}$ will give the network configuration of II, and hence all the minimal cuts of II are minimal-cuts of I.

$\square$ Theorem 2.1.

Theorem 2.2

With the notation of theorem 2.1, suppose that there is no set L'_f such that $L_f \subset L'_f$, and that all basic optimal solutions for L_f are also basic optimal solutions for L'_f. Then a minimal cut for L_f is $(X, \bar{X})$, where

$$X = \{(i \in N) \cup \{s\}/x_i \in L_f\} \ .$$

Proof

The proof will be carried out by contradiction. Clearly every node $\{i/x_i \in L_f\}$ belongs to X since there are no upper bounds on the variables y_i. Now, suppose that there exists a minimal cut $(X', \bar{X}')$ such that $X' = X \cup j$ where $x_j \notin L_f$. Then we can add to the network of problem II a new

non-basic variable $y_j \geq 0$ with value zero (i.e., a link connecting the source with node j with zero flow). By Theorem 2.1, all basic optimal solutions of II are also basic optimal solutions of I, where $L_f' = L_f \cup x_j$ in contradiction to the assumption that there exists no such set.

$\square$ Theorem 2.2

Corollary 2.1

Under the assumptions of Theorem 2.2, when maximal flow is achieved, all links:

$$\{(i,k) \in L / x_i \in L_f, \; x_k \notin L_f\} \; ,$$

have a flow equal to c_{ik} and all links:

$$\{(j,i) \in L / x_i \in L_f, \; x_j \notin L_f\} \; ,$$

have flow with value zero. (Follows from Theorem 2.2 and the Max-Flow-Min-Cut Theorem).

$\square$ Corollary 2.1

Now we are able to present a method for solving problem (2.8). Recall that step 1 of the algorithm calls for solving a series of linear programs: First the linear program corresponding to all states leaving the boundary, then those corresponding to all combinations of $(n-1)$ states leaving the boundary, etc. In other words, if L_f^1 and L_f^2 are two sets of states leaving the boundary corresponding to two successive substeps of step 1, then cardinality $L_f^2 \leq$ cardinality L_f^1. Now suppose that the current substep of step 1 corresponds to the set of states L_f. If in an earlier substep (say, corresponding to the set L_f') we obtained a basic optimal solution with $\{y_j = 0, \; \forall \, x_j \in L_f'/L_f\}$ and $L_f \subset L_f'$, then by Theorem 2.1 all optimal solutions of the current substep (L_f) are included in the set of optimal solutions of the earlier substep (L_f'). Therefore we do not need to solve the linear program corresponding to the current substep. In algorithmic

terms, there are two ways for checking the existence of such a set L_f':

1. To search all solutions obtained in earlier substeps corresponding to sets L_f' where $L_f \subset L_f'$, while looking for a solution with $\{y_j = 0 \ \forall \, x_j \in L_f'/L_f\}$.

2. To find the Maximal Flow corresponding to the current substep (L_f) and then to look for earlier substeps L_f' such that $L_f \subset L_f'$ with the same value of Maximal Flow.

Method (2) follows from the proof of Theorem 2.1.

Suppose now that there is no set L_f' such that $L_f \subset L_f'$ and such that the problem corresponding to L_f' has an optimal solution with $\{y_j = 0 \ \forall x_j \in L_f'/L_f\}$. In this case we apply to the network corresponding to L_f the Maximal Flow Algorithm. According to Corollary 2.1, when maximal flow is achieved, the network can be partitioned into two subnetworks as shown in Figure 2.4:
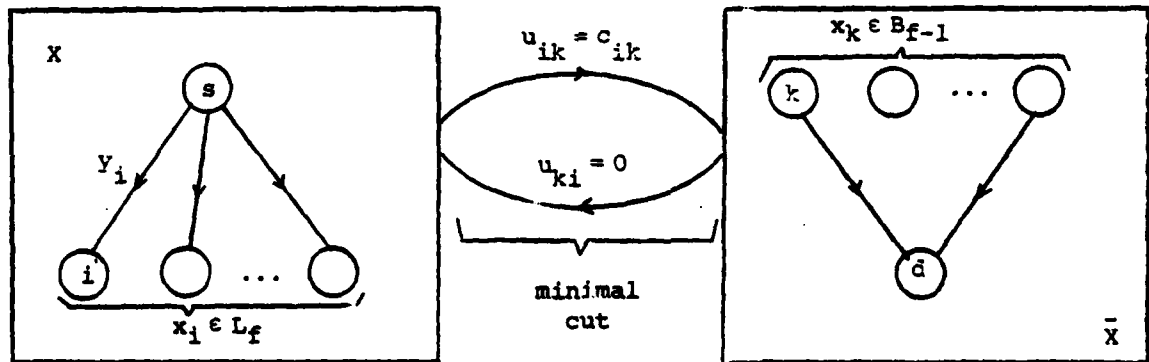
Figure 2.4 - The two subnetworks separated by a minimal cut.

Recall that since we are building a feedback control region, we are searching for all extremal values of $\{y_i/x_i \in L_f\}$ and the corresponding controls $\{u_{jk}/(j,k) \in L\}$. But we notice that only pivots on the subnetwork corresponding to X will lead to new extremal values of $\{y_j/x_i \in L_f\}$, so

that in order to obtain all optimal solutions to the problem corresponding to L_f we concentrate on the reduced network depicted in Figure 2.5:
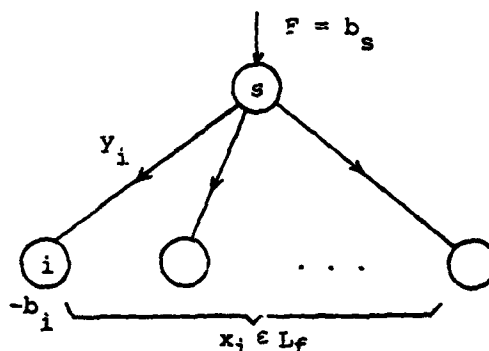
Figure 2.5 - Network configuration before the pivoting operation.

In Figure 2.5 we define:

$$b_i = \begin{cases} c_{id} + \sum_{\substack{j \in E(i) \\ x_j \notin L_f}} c_{ij} & \forall\ i \in I(d) \cap I(k/x_k \in B_{f-1}) \\[3ex] \sum_{\substack{j \in E(i) \\ x_j \notin L_f}} c_{ij} & \forall\ i \notin I(d)\ \text{ and }\ i \in I(k/x_k \in B_{f-1}) \\[3ex] c_{id} & \forall\ i \in I(d)\ \text{ and }\ i \notin I(k/x_k \in B_{f-1}) \\[2ex] 0 & \text{otherwise} \end{cases} \qquad (2.10)$$

$$b_s = \sum_{x_i \in L_f} b_i \qquad (2.10a)$$

For the first basic optimal solution we take

$$y_i = b_i \quad \forall\ x_i \in L_f$$

$$u_{ij} = 0 \quad \forall\ x_i \in L_f,\ j \in E(i) \cap \{i/x_i \in L_f\} \ . \qquad (2.11)$$

By pivoting on the network of Figure 2.5, we obtain all the required optimal solutions, when for each solution the flow values on the links in $\bar{X}$ are equal to those achieved when applying the Maximal Flow Algorithm.

Notice that at every substep of step 1 a further reduction of the network is obtained. Moreover, from considerations of Theorem 2.1, there might be many substeps for which there is no need to solve a linear program.

D. States Leaving the Boundary from Constructed Feedback Control Regions

We deal here with step 2 of the algorithm of Part A of this Section. In this step we find the leave-the-boundary values of the costates λ_i for all $x_i \in B_p$ corresponding to a constructed feedback control region R_p, and then we solve a linear program of the form (2.2) for each $L_p \subset B_p$. Step 2 is performed iteratively for each previously constructed feedback control region. We point out here, that the values of the costates corresponding to I_p and the values of the costates corresponding to states leaving the boundary (L_p) are positive. These details will be proved in Section 3. As for step 1 of the algorithm, step 2 can also be greatly simplified, as shown by the following theorem:

Theorem 2.2

Consider the problem

$$\text{Maximize} \quad \sum_{x_i \in I_{p-1}} a_i y_i \quad a_i > 0 \quad \forall \, x_i \in I_{p-1} \tag{2.12}$$

s.t.

$$-\sum_{j \in E(i)} u_{ij} + \sum_{k \in I(i)} u_{ki} + y_i = 0 \quad \forall \, x_i \in I_{p-1}$$

$$\sum_{j \in E(i)} u_{ij} - \sum_{k \in I(i)} u_{ki} = 0 \quad \forall \, x_i \in B_{p-1} \tag{2.12a}$$

$$y_i \geq 0 \quad \forall \, x_i \in I_{p-1}$$

$$\underline{u} \in U$$

A basic optimal solution of (2.12) with arbitrary coefficients $\{a_i\}$ is also a basic optimal solution of (2.12), with $\{a_i = 1, \forall \, x_i \in I_{p-1}\}$.

Proof

Assume that:

F_1 = optimal value of (2.12) with $\{a_i = 1, \ \forall x_i \in I_{p-1}\}$.

$\underline{y}^*$ = optimal solution vector to (2.12) with $\{a_i > 0, \ \forall x_i \in I_{p-1}\}$.

Now suppose that:

$$\sum_{x_i \in I_{p-1}} y_i^* < F_1 \quad . \tag{2.13}$$

Then the Maximal Flow Theorem (see Appendix A) implies that if (2.13) is satisfied, we can always find a path between the source and the destination nodes of the network corresponding to problem (2.12) on which we can increase the flow, in contradiction with the assumption that $\underline{y}^*$ is an optimal solution to (2.12) with $\{a_i > 0, \ \forall x_i \in I_{p-1}\}$. Now, noting that:

$$\sum_{x_i \in I_{p-1}} y_i^* > F_1 \quad , \tag{2.14}$$

cannot be satisfied, since clearly a flow satisfying (2.14) violates the constraints (2.12a) by the Maximal Flow Theorem, then:

$$\sum_{x_i \in I_{p-1}} y_i^* = F_1 \quad . \tag{2.15}$$

Moreover, since the structure of the network corresponding to problem (2.12) does not depend on the weightings $\{a_i\}$, then all __basic__ optimal solutions of (2.12) with $\{a_i > 0, \ \forall x_i \in I_{p-1}\}$ are also __basic__ optimal solutions of (2.12) with $\{a_i = 1, \ \forall x_i \in I_{p-1}\}$.

$\square$ Theorem 2.2

Now, transforming the linear program (2.2) into a Maximal Flow Problem in a network with weights on the links connecting the source with every node, we have:

$$\text{Maximize} \quad \sum_{x_i \epsilon I_{p-1}} \lambda_i y_i \tag{2.16}$$

s.t.

$$-\sum_{j \epsilon E(i)} u_{ij} + \sum_{k \epsilon I(i)} u_{ki} + y_i = 0 \quad \forall x_i \epsilon I_{p-1}$$

$$\sum_{j \epsilon E(i)} u_{ij} - \sum_{k \epsilon I(i)} u_{ki} = 0 \quad \forall x_i \epsilon B_{p-1} \tag{2.16a}$$

$$y_i \geq 0 \quad \forall x_i \epsilon I_{p-1}$$

$$\underline{u} \epsilon U$$

According to Theorem 2.2, all solutions of problem (2.16) are solutions of problem (2.9) with $L_f = I_{p-1}$, therefore every linear program of step 2 of the algorithm of Part A of this section reduces to the following simple problem:

From among all solutions of problem (2.9) with $L_f = I_{p-1}$, (denoted by $\underline{y}^*$) choose those satisfying:

$$\underset{\underline{y} \epsilon \{\underline{y}^*\}}{\text{Max}} \sum_{x_i \epsilon I_{p-1}} \lambda_i y_i \quad .$$

Note that step 1 is performed for all possible combinations of states, assuring that we can always find a set L_f such that $L_f = I_{p-1}$.

Section 3

THEORETICAL RESULTS

A. The Structure of the y-Constraint-Figure
in the Positive Orthant of the y-Space

In [M3] the y-constraint-Figure, Y, is defined as follows:

$$Y \stackrel{\Delta}{=} \{ \underline{y} \in \mathbf{R}^n / \underline{u} \in U \}$$

where

$$U = \begin{cases} 0 \leq u_{ik} \leq c_{ik} \\ \\ \forall (i,k) \in L \end{cases}$$

and the linear transformation relating Y and U is given by:

$$\underline{y}(\tau) \stackrel{\Delta}{=} -\dot{\underline{x}}(\tau) = -B\underline{u}(\tau) \,,$$

where B is the incidence matrix of the network.

Since U is a bounded convex polyhedron in $\mathbf{R}^m$, its image Y is clearly a bounded convex polyhedron in $\mathbf{R}^n$. Thus, every face of Y can be analytically described by an expression of the form:

$$\sum_{i=1}^{n} a_i y_i = \ell(\{a_i\}) \,, \tag{3.1}$$

where $\{a_i\}$ is a set of coefficients and $\ell(\{a_i\})$ a constant with value depending on the set $\{a_i\}$.

Our first aim is to prove that all coefficients $\{a_i\}$ corresponding to any face of Y that is in the positive orthant of the y-space are nonnegative. This is easily demonstrated if we consider a constraint of the form

$$\sum_{i=1}^{n} a_i y_i \leq \ell(\{a_i\}) \,, \tag{3.2}$$

passing through the positive orthant of the $\underline{y}$-space so that there is at least

a point $\underline{y}^1$ lying on the boundary of (3.2), satisfying:

$$y_i^1 > 0 \quad \forall \, i \in N \; . \tag{3.3}$$

Suppose now that a coefficient of (3.2), say a_k, is negative and let us con-

centrate on one of the paths carrying flow from the node k to the destination.

At least one such a path exists since $y_k^1 > 0$ by (3.3). Clearly, by decreas-

ing the flow on this path the flow on the network will remain feasible, but we

violate the constraint (3.2) since a_k is negative. Therefore all coeffici-

ents of (3.2) must be nonnegative. Moreover, clearly the constant $\ell(\{a_i\})$

is given by:

$$\ell(\{a_i\}) = \operatorname*{Max}_{y} \sum_{i=1}^{n} a_i y_i \; . \tag{3.4}$$

In order to obtain an explicit set of constraints defining Y in the positive

orthant of the $\underline{y}$-space we must take all the constraints (3.2) and eliminate

those which are redundant. To do this, we shall present now three redundancy

conditions and we will prove that if a constraint such as (3.2) satisfies at

least one of the conditions, it is redundant. Moreover we shall prove that

a constraint satisfying not one of the above conditions is not redundant. In

other words, a necessary and sufficient condition for a constraint of the form

(3.2) to be redundant is the fulfilment of at least one of the conditions.

For a constraint such as (3.2) we denote

$$D = \{i/0 < a_i \in \{a_i\}\}$$

and

$$k(D) = \operatorname*{Max}_{y} \sum_{D} y_i \qquad .$$

Redundancy Condition 1

The hyperplane

$$\sum_D a_i y_i = \ell(\{a_i\}) \tag{3.5}$$

is redundant if at least two of its coefficients are different.

Proof

By Theorem 2.3:

$$\{\underline{y} \in \mathcal{V} / \sum_D a_i y_i = \ell(\{a_i\})\} \subset \{\underline{y} \in \mathcal{V} / \sum_D y_i = k(D)\} \quad .$$

Namely, to a hyperplane of the form (3.5) always corresponds another hyperplane with unity coefficients containing all points of tangency of (3.5) with $\mathcal{V}$.

Redundancy Condition 2

The hyperplane

$$\sum_D y_i = k(D) \tag{3.6}$$

is redundant if there exists a set of indeces D' such that:

$$D \subset D' \tag{3.7a}$$

with

$$\sum_{D'} y_i = k(D) \quad . \tag{3.7b}$$

Proof

By Theorem 2.1, if (3.7) is satisfied, then:

$$\{\underline{y} \in \mathcal{V} / \sum_D y_i = k(D)\} \subset \{\underline{y} \in \mathcal{V} / \sum_{D'} y_i = k(D)\} \quad .$$

Namely, the hyperplane (3.7b) contains all points of tangency of (3.6) with $\mathcal{V}$.

Redundancy Condition 3

The hyperplane

$$\sum_D y_i = k(D) \tag{3.8}$$

is redundant if there exist sets of indeces $\{B_j, \; j = 1,2,\ldots,r\}$, satisfying:

$$\sum_{j=1}^{r} k(B_j) = k(D) \quad , \tag{3.9a}$$

$$\left.\begin{aligned}
&\bigcup_{j=1}^{r} B_j = D \\[2ex]
&B_{j_1} \cap B_{j_2} = \{\phi\} \;\; \forall j_1, j_2 \in \{1,2,\ldots,r\}
\end{aligned}\right\} \tag{3.9b}$$

where

$$k(B_j) = \underset{y}{\text{Max}} \sum_{B_j} y_i \quad . \tag{3.9c}$$

Proof

There are no common links to the minimal cuts corresponding to the hyperplanes (3.9c), because the existence of one such link implies:

$$k(D) = \underset{y}{\text{Max}} \sum_{D} y_i \;\; = \;\; \underset{y}{\text{Max}} \sum_{j=1}^{r} \sum_{B_j} y_i \;\; < \;\; \sum_{j=1}^{r} \underset{y}{\text{Max}} \sum_{B_j} y_i \;\; = \;\; \sum_{j=1}^{r} k(B_j)$$

or $\;\; k(D) < \sum\limits_{j=1}^{r} k(B_j) \;\;$ contradicting (3.9a).

Now if there are no common links to the minimal cuts corresponding the hyperplanes (3.9c), then by (3.9b) follows that:

$$\{\underline{y} \in Y / \sum_{D} y_i = k(D)\} \equiv \{\underline{y} \in Y / \underline{y} \in \bigcap_{j=1}^{r} (\sum_{B_j} y_i = k(B_j))\} \; ,$$

therefore (3.8) is redundant.

$\square$ Redundancy Conditions

We prove now by contradiction that a hyperplane of the form

$$\sum_{D} y_i = k(D) \tag{3.10}$$

satisfying none of the three redundancy conditions is not redundant. To this end we assume that (3.10) does not satisfy redundancy conditions 2 and 3

(condition 1 is clearly not satisfied by (3.10), but is redundant. If (3.10) is redundant, then there is a non-redundant hyperplane of the form:

$$\sum_{D'} y_i = k(D') \tag{3.11}$$

such that:

$$\{y \in Y / \sum_{D} y_i = k(D)\} \subset \{y \in Y / \sum_{D'} y_i = k(D')\} \quad , \tag{3.12}$$

and this for the following reasons:

Every solution of (3.10) over Y is feasible and tangent to Y. The surface of tangency common to (3.10) and Y is clearly a convex hyperplane of dimension m, where $m < (n-1)$, (if $m = (n-1)$ then (3.10) is not redundant). Now, an m-dimensional hyperplane on the boundary of Y corresponds to the intersection of $(n-m)$ non-redundant hyperplanes of dimension $(n-1)$, and every point of tangency of (3.10) with Y belongs to every one of these $(n-m)$ hyperplanes. Hence, (3.12) applies to every one of the hyperplanes forming the surface of tangency, and for the proof we need to consider only one of them, say the hyperplane given by (3.11).

Now, we prove that (3.10) is not redundant. Note that in general the index sets D, D' can be partitioned as follows:

$$D = \{D_1, D_2\}$$

$$D' = \{D_1, D_3\}$$

where

$$D_1 \cap D_2 = D_1 \cap D_3 = D_2 \cap D_3 = \{\phi\} \quad .$$

Our aim is to contradict (3.12), considering all possible relations between the sets D and D'. There are four cases:

__Case 1:__ $D_1 = \{\phi\}$; $D_2 \neq \{\phi\}$; $D_3 \neq \{\phi\}$.

In this case we choose a point y^* satisfying:

$$\sum_D y_i^* = \sum_{D_2} y_i^* = k(D) \quad,$$

where

$$y_i^* > 0 \quad \forall i \in D_2$$

$$y_i^* = 0 \quad \forall i \in D_3 \quad.$$

Clearly, since $D_3 \neq \{\phi\}$ we have:

$$\sum_{D'} y_i^* = \sum_{D_3} y_i^* = 0 < k(D') \quad,$$

thus contradicting (3.12).

__Case 2:__ $D_3 = \{\phi\}$; $D_1 \neq \{\phi\}$; $D_2 \neq \{\phi\}$.

Here we choose a point y^* satisfying:

$$\sum_D y_i^* = k(D)$$

$$\sum_{D_2} y_i^* = k(D_2) \quad.$$

(3.13)

But if (3.10) does not satisfy redundancy condition 3, then:

$$k(D) < k(D_1) + k(D_2) \quad,$$

(3.14)

therefore from (3.13) and (3.14) follows that:

$$\sum_{D'} y_i^* = \sum_{D_1} y_i^* = k(D) - k(D_2) < k(D_1) = k(D') \quad,$$

or

$$\sum_{D'} y_i^* < k(D') \quad,$$

in contradiction to (3.12).

Case 3

Let us choose a point $\underline{y}^*$ satisfying $\sum_D y_i^* = k(D) = k(D_1)$,

where

$$y_i^* > 0 \quad \forall i \in D_1 = D$$

$$y_i^* = 0 \quad \forall i \in D_3$$

Now, if $\sum_{D'} y_i^* = k(D') = k(D_1)$, then since $D \subset D'$, hyperplane (3.10)

satisfies redundancy condition 2, contradicting the assumptions, thus.

$$\sum_{D'} y_i^* < k(D') \quad ,$$

in contradiction to (3.12).

Case 4: $D_1 \neq \{\phi\}; \quad D_2 \neq \{\phi\}; \quad D_3 \neq \{\phi\}$.

In this case we choose a point satisfying:

$$\sum_D y_i^* = k(D)$$

where

$$y_i^* > 0 \quad \forall i \in D \qquad . \tag{3.15}$$

$$y_i^* = 0 \quad \forall i \in D_3$$

This point satisfies $\sum_{D'} y_i^* < k(D')$ since if

$$\sum_{D'} y_i^* = k(D') \quad , \tag{3.16}$$

then from (3.15) and (3.16) follows that:

$$\underset{Y}{\text{Max}} \sum_{D \cup D'} y_i = \sum_{D' \cup D} y_i^* = k(D) \quad .$$

Thus, (3.10) satisfies redundancy condition 2 in contradiction to the assumptions.

Therefore $\underline{y}^*$ contradicts (3.12).

In order to illustrate the procedure for obtaining the $\underline{y}$-constraint

figure in the positive orthant of the $\underline{y}$-space, we present the following simple

example in two dimensions:

Example 3.1

Consider the network depicted in Figure 3.1:
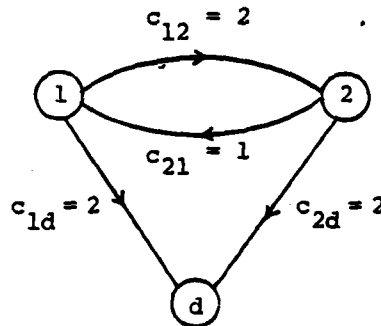
<u>Figure 3.1</u> - Network of Example 3.1

Finding the maximal flow values of the networks shown in Figure 3.2, we obtain:

(a) $\text{Max}_y \, y_1 = 4$ (b) $\text{Max}_y \, y_2 = 3$ (c) $\text{Max}_y \, (y_1 + y_2) = 4$
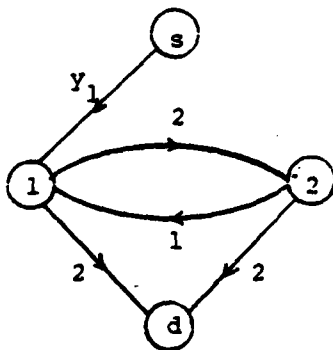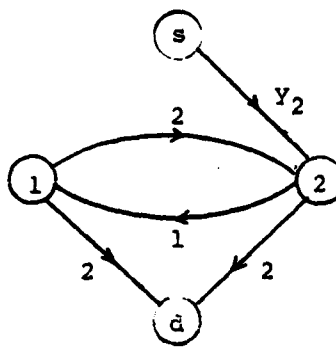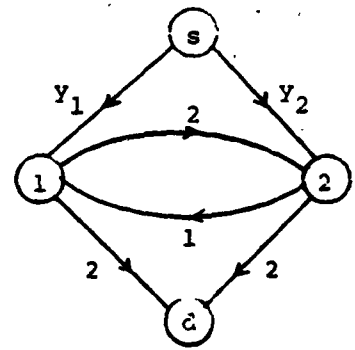
(a)   (b)   (c)

<u>Figure 3.2</u> - Networks to find the maximal flow.

Clearly the hyperplane $y_1 = 4$ is redundant by redundancy condition 2, and Y is defined by

$$y_2 = 3$$

and

$$y_1 + y_2 = 4 \quad .$$

The region Y in the positive orthant of the y-space is depicted in Figure 3.3:
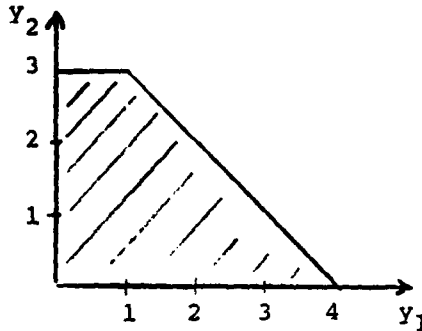
Figure 3.3 - Structure of Y in Example 3.1

☐ Example 3.1

B. Special Properties of the Feedback Solution for Single Destination Networks with all Unity Weightings in the Cost Functional

As we mentioned in Part A of Section 2, several features associated with the Constructive Dynamic Programming Algorithm in the case of multiple destination networks, or when the cost functional has nonequal weightings, complicate the formulation of a computational scheme to implement the algorithm. However, in the case of single destination networks with all unity weightings in the cost functional various simplifications result, thus permitting us to develop a compact algorithm for building the feedback solution. These simplifications are stated and proved in [M1]. However, the approach in this work permits us to carry out the proofs in a more simple and straightforward manner.

We begin with the central theorem of this section. The proof of this theorem is the basis of all subsequent proofs.

Theorem 3.1

The value of the leave-the-boundary costates are unique at a given boundary junction time t_p .

Proof

Assume that t_p is a fixed boundary junction time and consider first the case in which we allow all states lying on the boundary to leave it, that is $L_p = B_p$. Let us denote by t_p^+ the time just before the states in B_p leave the boundary, and by t_p^- the time just after the states in B_p have left the boundary, (going backwards in time). Now, the restricted Hamiltonian at t_p^+ is:

$$H(t_p^+): \quad z(t_p^+) = \sum_{x_i \epsilon I_p} \lambda_i(t_p^+) y_i \quad , \qquad (3.17)$$

and the set of operating points in the $\underline{y}$-space is given by:

$$Y_p = \{ \underline{y} \epsilon Y / z(t_p^+) = \underset{y}{\text{Max}} \sum_{x_i \epsilon I_p} \lambda_i(t_p^+) y_i \} \quad .$$

We assume in this proof that all operating points in Y_p are nonnegative, that is, they are in the positive orthant of the $\underline{y}$-space. Later we shall prove that in the case of single destination networks with all unity weightings in the cost functional, we can consider only this orthant and that there always exists a solution there.

We denote by "a" the optimal value of the restricted Hamiltonian (3.17), that is:

$$\underset{y}{\text{Max}} \sum_{x_i \epsilon I_p} y_i = a \quad . \qquad (3.18)$$

By the Constructive Dynamic Programming Algorithm, we have to find the leave-the-boundary costate values associated with all states in B_p which allow for the optimal departure of the states from the boundary at t_p, (see [M2]). In geometrical terms, the above step calls for finding the values of $\{\lambda_j(t_p^-), \forall x_j \epsilon B_p \}$, for which the global Hamiltonian

$$H(t_p^-): \quad z(t_p^-) = \sum_{x_i \epsilon I_p} \lambda_i(t_p^+) y_i + \sum_{x_i \epsilon B_p} \lambda_i(t_p^-) y_i \quad , \qquad (3.19)$$

contains a B_p-positive face of Y (see [M3]). Notice that since the operation

is made by rotating $H(t_p^-)$ around $H(t_p^+)$, then $Y_p \in H(t_p^-)$, therefore:

$$\underset{y}{\text{Max}} \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \in B_p} \lambda_i(t_p^-)y_i = \underset{y}{\text{Max}} \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i = a \quad . \qquad (3.20)$$

Now we prove the assertion:

Assertion 3.1

(i) $\lambda_i(t_p^+) \geq 0 \quad \forall x_i \in I_p$

(ii) $\lambda_i(t_p^-) \geq 0 \quad \forall x_i \in B_p$.

Proof

(i) If $x_i \in I_p$, then x_i has left the boundary for the last time at τ, where $\tau > t_p$ with an optimal slope $y_i > 0$. Now, clearly $\lambda_i(\tau) \geq 0$ since otherwise it is not optimal for x_i to leave the boundary at τ. Therefore, since $\dot{\lambda}_i(t) = -1, \forall t \in (t_p, \tau)$, we have that $\lambda_i(t_p^+) > 0$, $\forall x_i \in I_p$.

(ii) If $\lambda_i(t_p^-) < 0$ for some $x_i \in B_p$, then it is not optimal for x_i to leave the boundary, but x_i does leave the boundary, so that $\lambda_i(t_p^-) \geq 0, \forall x_i \in B_p$.

$\square$ Assertion 3.1.

Now, since $\{\lambda_i(t_p^+) > 0, \forall x_i \in I_p\}$, then by Theorem 2.3 we have that:

$$\{y \in Y/ \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i = a\} \subset \{y \in Y/ \sum_{x_i \in I_p} y_i = k(I_p)\} \quad , \qquad (3.21)$$

where

$$k(I_p) = \underset{y}{\text{Max}} \sum_{x_i \in I_p} y_i \quad .$$

Moreover, since $\{\lambda_i(t_p^-) \geq 0, \forall x_i \in B_p\}$, then by Theorem 2.3 it follows that:

$$\{\underline{y}\epsilon Y / \sum_{x_i\epsilon I_p} \lambda_i(t_p^+)y_i + \sum_{x_i\epsilon B_p} \lambda_i(t_p^-)y_i = a\} \subset \{\underline{y}\epsilon Y / \sum_{x_i\epsilon I_p} y_i + \sum_{x_i\epsilon A} y_i = k(I_p\cup A)\},$$

$$(3.22)$$

where A is a set of states such that $A\subset B_p$. Clearly, since $Y_p\epsilon H(t_p^-)$, then by (3.21) and (3.22), Y_p belongs to all sets in (3.21) and (3.22), therefore:

$$\underset{y}{\text{Max}}\{\sum_{x_i\epsilon I_p} y_i + \sum_{x_i\epsilon A} y_i\} = \underset{y}{\text{Max}}\sum_{x_i\epsilon I_p} y_i = k(I_p) \quad . \qquad (3.23)$$

Next, we prove the following assertion:

Assertion 3.2

There is a (unique) maximal set A satisfying (3.23).

Proof:

Suppose there exist m sets $\{A_j, \quad j = 1,2,\ldots,m\}$, satisfying:

$$\underset{y}{\text{Max}}\{\sum_{x_i\epsilon I_p} y_i + \sum_{x_i\epsilon A_j} y_i\} = k(I_p), \quad j = 1,2,\ldots,m \quad .$$

Now, consider the following networks:

(a) Network having links connecting the source with every node in I_p, namely, network corresponding to the problem $\underset{y}{\text{Max}}\sum_{x_i\epsilon I_p} y_i$, (see parts B and C of Section 2).

(b) Network having links connecting the source with every node in I_p and in A_1, namely, network corresponding to the problem
$$\underset{y}{\text{Max}}\{\sum_{x_i\epsilon I_p} y_i + \sum_{x_i\epsilon A_1} y_i\}.$$

(c) As (b), but A_1 being replaced by $A_2,A_3,\ldots,A_m$.

Now by Theorem 2.1, all networks in (a), (b) and (c) have at least one common minimal cut. But this minimal cut is obviously a minimal cut of the network corresponding to the problem:

$$\underset{y}{\text{Max}}\{ \sum_{x_i \in I_p} y_i + \sum_{\substack{m \\ x_i \in \bigcup_{j=1} A_j}} y_i \}$$

so that

$$\underset{y}{\text{Max}}\{ \sum_{x_i \in I_p} y_i + \sum_{\substack{m \\ x_i \in \bigcup_{j=1} A_j}} y_i \} = k(I_p) \ .$$

Therefore, since there is a finite number of states, then there is a (unique) maximal set $A = \bigcup_{j=1}^{m} A_j$ satisfying (3.23).

$\square$ Assertion 3.2

From now on, the face of V (of dimension less or equal to $n-1$), defined by the set $I_p \cup A$ will be referred to as the $(I_p \cup A)$-face.

Now, we concentrate on the set of states B_p and we prove the following assertion:

Assertion 3.3

If $x_j \in B_p$ but $x_j \notin A$, then $\lambda_j(t_p^-) = 0$.

Proof:

The proof will be carried by contradiction. Assume that $\lambda_j(t_p^-) = 0$ for a set of states D such that $D \subset B_p/A$. Then, the enlarged Hamiltonian is given by:

$$H(t_p^-): \quad z(t_p^-) = \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \in A} \lambda_i(t_p^-)y_i + \sum_{x_j \in D} \lambda_j(t_p^-)y_j \tag{3.24}$$

Moreover, from Theorem 2.3 follows that:

$$\{y \in V/z(t_p^-) = a\} \subset \{y \in V/ \sum_{x_i \in I_p} y_i + \sum_{x_i \in A} y_i + \sum_{x_j \in D} y_j = k(I_p \cup A \cup D)\} \ . \tag{3.25}$$

Now, if $k(I_p \cup A \cup D) > k(I_p)$, then $Y_p \notin H(t_p^-)$ which is a contradiction; and if $k(I_p \cup A \cup D) = k(I_p)$, then clearly $D \subset A$ since A is the maximal set

satisfying (3.23), again a contradiction. Therefore $\{\lambda_j(t_p^-) = 0, \; \forall \; x_j \in B_p/A\}$.

$\Box$ Assertion 3.3.

From assertion 3.3 follows that the enlarged Hamiltonian is given by:

$$H(t_p^-): \quad z(t_p^-) = \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \in A} \lambda_i(t_p^-)y_i = a \; , \qquad (3.26)$$

and that the B_p-positive-face is given by the intersection of the Hamiltonian (3.26) with the $(I_p \cup A)$-face (3.23).

Now we prove by contradiction, that the Hamiltonian (3.26) is unique, implying that the B_p-positive-face is unique. Suppose that there are two Hamiltonians of the form (3.26) such that the intersection of every one of them with the $(I_p \cup A)$-face gives two distinct B_p-positive-faces. These Hamiltonians are:

$$z^1(t_p^-) = \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \in A} \lambda_i^1(t_p^-)y_i = a \; , \qquad (3.27a)$$

$$z^2(t_p^-) = \sum_{x_i \in I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \in A} \lambda_i^2(t_p^-)y_i = a \; , \qquad (3.27b)$$

where for the two Hamiltonians to be distinct, there is at least one $x_i \in A$ such that $\lambda_i^1(t_p^-) \neq \lambda_i^2(t_p^-)$. This also implies that there exist at least two points $\underline{y}^1 > 0$ and $\underline{y}^2 > 0$ of the $(I_p \cup A)$-face, such that:

$$\left. \begin{array}{l} z^1(t_p^-) = a \\[2mm] z^2(t_p^-) < a \end{array} \right\} \text{ at } \underline{y}^1 \; , \qquad (3.28a)$$

and

$$\left. \begin{array}{l} z^2(t_p^-) = a \\[2mm] z^1(t_p^-) < a \end{array} \right\} \text{ at } \underline{y}^2 \; . \qquad (3.28b)$$

Now consider the intersection of one of the Hamiltonians, say (3.28a) with the $(I_p \cup A)$-face. Since both, the Hamiltonian and the $(I_p \cup A)$-face are convex and linear, their intersection will give a convex polytope. Moreover, since the $(I_p \cup A)$-face is bounded, the intersection will give a convex polyhedron. Thus, starting at an extremal point of the polyhedron, we can always reach any other extremal point of the polyhedron through a series of pivots. Note that all extremal points of Y_p are contained in the above intersection. Suppose that, starting at some extremal point of Y_p, we want to reach the point $\underline{y}^1$ through a series of pivots in such a way that after every pivot of the above series we reach an extremal point of the polyhedron, that is, satisfying the equations of the Hamiltonian and the $(I_p \cup A)$-face. We state that the transition between Y_p and $\underline{y}^1$ can be done in the following way:

(i) Starting at Y_p, reach the extremal point $\underline{y}^a$ through a series of pivots on the polyhedron, where

$$\left.\begin{aligned} y_i^a &= y_i^1 \quad \forall\, x_i \in I_p \cup A \\[2mm] y_i^a &= 0 \quad \forall\, x_i \in B_p / A \end{aligned}\right\} \; . \tag{3.29}$$

(ii) Without changing the coordinates corresponding to the states in $I_p \cup A$, reach $\underline{y}^1$.

The above procedure is justified by noting that if we are moving over the $(I_p \cup A)$-face, then every point reached after a pivot operation must satisfy:

$$\sum_{x_i \in I_p} y_i + \sum_{x_i \in A} y_i = k(I_p) \; , \tag{3.30}$$

and by Corollary 2.1 the network looks as in Figure 3.4:
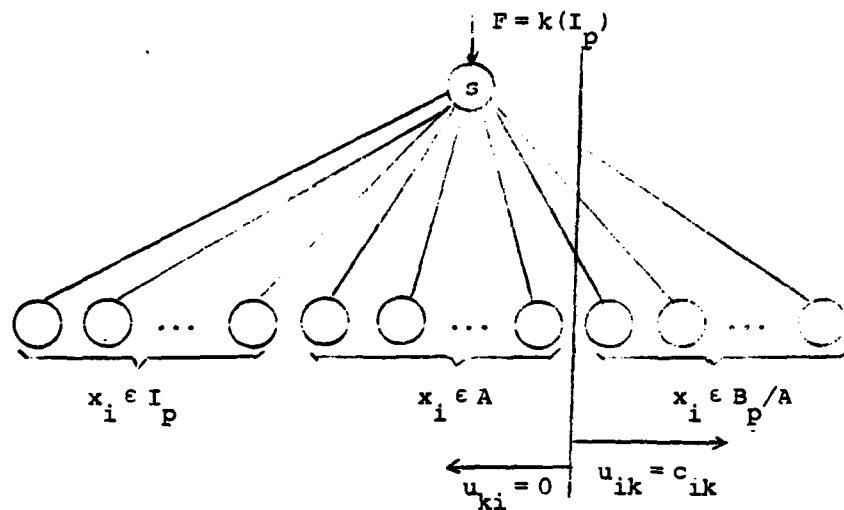
Figure 3.4 - Network reaching the point $\underline{y}^1$.

Clearly, in order to reach $\underline{y}^1$ we cannot make pivots between states in $I_p \cup A$ and B_p/A because by making such a pivot we violate (3.30).

Recall that our aim is to contradict (3.27), that is to prove that the Hamiltonian is unique. To this end we shall deal with pivots in the part of the network corresponding to states in $I_p \cup A$. First we prove the following assertion:

Assertion 3.4

Starting at Y_p, we can always find a path in the network between each node $\{j/x_j \in A\}$ and some node $\{i/x_i \in I_p\}$, that permits us to perform a pivot while increasing (from zero) the flow in the link corresponding to y_j and still to remain on the Hamiltonian (3.27a).

Proof

Consider the series of pivots that reaches $\underline{y}^1$ from Y_p. Suppose we show that after completing n pivots of this series, we could also have increased the flow y_j, where $x_j \in A$, corresponding to the $(n+1)$-st pivot before the n-th pivot, reaching in this way a point $\underline{y}^b$ that satisfies the equation of the Hamiltonian (3.27a). Clearly, the application of the same

arguments to the new series of pivots that reaches $\underline{y}^b$ from Y_p provides us with the induction step that proves our assertion.

For the sake of clarity the following notation will be used:

$y_1^{(j)}$ = the slack variable y_j that increases (from zero) in the i-th pivot.

$y_2^{(i)}$ = the slack variable that decreases its flow in the i-th pivot.

$\lambda^{(i)}$ = the costates corresponding to the slack variables of the i-th pivot.

$\underline{y}(n)$ = the point reached after the n-th pivot.

Δy = the increase of flow in a slack link after pivoting.

Suppose we cannot perform the (n+1)-st pivot before the n-th one. Clearly, this occurs only when there is at least one link common to the n-th and (n+1)-st pivot cycles, such that the execution of the (n+1)-st pivot is possible only after the n-th one. All such "critical" links must be in opposite directions on the n-th and the (n+1)-th pivot cycles. We shall contradict our assumption by showing that:

(a) $\lambda^{(n+1)} = \lambda^{(n)}$.

(b) before the n-th pivot there exists a cycle that permits increasing $y_1^{(n+1)}$ while decreasing $y_2^{(n)}$.

Consider first the situation <u>before the n-th pivot</u>, (see Figure 3.5). Suppose that (k,j) is the first critical link on the (n+1)-st pivot cycle. Clearly, since the link is critical it belongs also to the n-th pivot cycle. Then follow the (n+1)-st pivot cycle from $y_1^{(n+1)}$ up to the node k and then the n-th pivot cycle up to $y_2^{(n)}$. Clearly we can perform a pivot on this cycle. Moreover, $\lambda^{(n+1)} \leq \lambda^{(n)}$ because if $\lambda^{(n+1)} > \lambda^{(n)}$ then, performing the pivot corresponding to this cycle will give:

$$\sum_{x_i \epsilon I_p} \lambda_i(t_p^+)y_i(n-1) + \sum_{x_i \epsilon A} \lambda_i(t_p^-)y_i(n-1) + \lambda^{(n+1)}\Delta y + \lambda^{(n)}\Delta y = a + (\lambda^{(n+1)} - \lambda^{(n)})\Delta y \cdot a,$$

thus, contradicting the optimality of the Hamiltonian (3.27a).

Next, consider the situation <u>after the n-th pivot,</u> (see Figure 3.5).
Let (k',j') be the last critical link on the $(n+1)$-st pivot cycle. Consider
the following cycle: start at $y_2^{(n)}$, follow the n-th pivot cycle backwards
up to the node k' and from there on follow the $(n+1)$-st pivot cycle (for-
wards), up to $y_2^{(n+1)}$. Clearly we can perform a pivot on this cycle and hence
$\lambda^{(n)} \leq \lambda^{(n+1)}$ for the same reasons as before.

—— n-th pivot cycle links; - - - $(n+1)$-st pivot cycle links; ... critical links
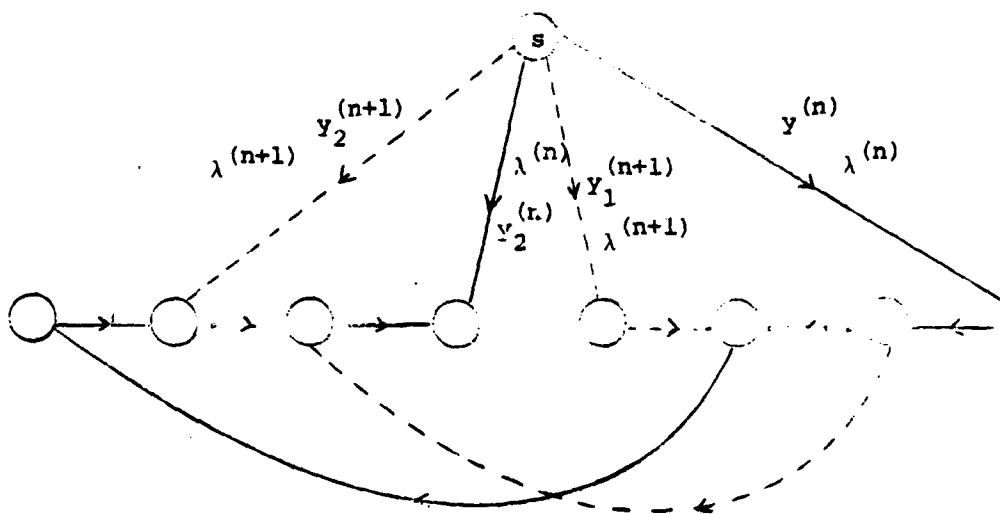
<u>Figure 3.5</u> - Example of critical links.

Therefore we conclude, that $\lambda^{(n)} = \lambda^{(n+1)}$ and as said already, before
the n-th pivot, there is a cycle including $y_1^{(n+1)}$ and $y_2^{(n)}$ on which a pivot
can be performed so that we can increase $y_1^{(n+1)}$ (from zero) before the n-th
pivot while reaching a point that satisfies the equation of the Hamiltonian
(3.27a). This proves the induction step.

□ Assertion 3.4

Now we contradict (3.27). By assertion 3.4, starting at Y_p, we can reach points y^* by performing a single pivot on the network. These points lie on the $(I_p \cup A)$-face and also satisfy the Hamiltonian equation (3.27a), namely:

$$\sum_{x_i \epsilon I_p} \lambda_i(t_p^+)y_i^* + \lambda_m^1(t_p^-)y_m^* = a, \quad \forall x_m \epsilon A \quad . \tag{3.31}$$

We check now the second Hamiltonian (3.27b) at the above points. Notice that since we assumed that (3.27b) is optimal over Y, it must satisfy:

$$\sum_{x_i \epsilon I_p} \lambda_i(t_p^+)y_i^* + \lambda_m^2(t_p^-)y_m^* \leq a, \quad \forall x_m \epsilon A \quad . \tag{3.32}$$

From (3.31) and (3.32) follows that:

$$\lambda_m^2(t_p^-) \leq \lambda_m^1(t_p^-), \quad \forall x_m \epsilon A \quad . \tag{3.33}$$

But obviously, assertion 3.4 holds also for the second Hamiltonian (3.27b), hence:

$$\lambda_m^1(t_p^-) \leq \lambda_m^2(t_p^-), \quad \forall x_m \epsilon A \quad . \tag{3.34}$$

Thus, from (3.33) and (3.34), we have:

$$\lambda_m^1(t_p^-) = \lambda_m^2(t_p^-), \quad \forall x_m \epsilon A \quad .$$

We then conclude that the B_p-positive-face is unique and therefore the leave-the-boundary costates are unique.

When only some of the states leave the boundary, we consider the restricted Hamiltonian:

$$H_{p-1}(t_p^-): \quad z(t_p^-) = \sum_{x_i \epsilon I_p} \lambda_i(t_p^+)y_i + \sum_{x_i \epsilon L_p} \lambda_i(t_p^-)y_i \quad ,$$

with $L_p \subset B_p$. But in order to satisfy the necessary conditions the optimization must be global, namely:

$$H_{p-1}(t_p^-) = H(t_p^-) \cap R^{\sigma_p + \rho_p} \quad ,$$

where

$$\sigma_p \triangleq \text{cardinality of } I_p$$

$$\rho_p \triangleq \text{cardinality of } L_p \quad ,$$

and the basis vectors of $\mathbb{R}^{\sigma_p + \rho_p}$ are the elements of I_{p-1}. Therefore the values of the leave-the-boundary costates in the case of $L_p \subseteq B_p$ are identical to those corresponding to the case of B_p so that they are unique.

$$\square \quad \text{Theorem 3.1.}$$

Theorem 3.2

If the values of the costates corresponding to states travelling on boundary arcs during the interval between two successive junction times t_{p+1} and t_p are equal to the leave-the-boundary costate values, then every costate satisfies exactly one of the following conditions:

$$\dot{\lambda}_i(\tau) = \lambda_i(\tau) = 0, \qquad \forall \tau \in (t_p, t_{p+1}) \quad , \qquad (3.35a)$$

$$\dot{\lambda}_i(\tau) = -1; \quad \lambda_i(\tau) > 0, \quad \forall \tau \in (-\infty, t_{p+1}) \quad . \qquad (3.35b)$$

Moreover, in every feedback control region there always exists an optimal control satisfying $y \geq 0$.

Proof

Every optimal trajectory in the state space is characterized by a control switch time sequence $(t_f, t_{f-1}, \ldots, t_{p+1}, t_p, t_{p-1}, \ldots)$, when each switch time represents a transition between two neighboring feedback control regions in the state space (see [M1]). The interval between two successive switch times t_{p+1} and t_p is characterized by the set of states travelling on interior arcs, I_p. Considering every possible control switch time sequence insures that we have traversed every feedback control region, thus covering the whole state space. To prove the Theorem we shall consider a typical control switch time sequence, and we will prove by induction that the costate trajectories satisfy (3.35), and also that there exists

at least one optimal control for which $y \geq 0$ within every feedback control region. Notice that the existence of such an optimal control allows us to consider only the set of trajectories for which there is no return to the boundary of states $(x = 0)$, backwards in time.

Now we proceed with the proof. Consider first the interval (t_p, t_{p-1}). Since $\{\lambda_i(t_f) = 0, \forall x_i \in L_f\}$ (see [M2], pp. 25-26), at t_f^- all costates corresponding to the states in L_f are identical. Now, by Theorem 3.1, for the costates corresponding to states on the boundary to be at their leave-the-boundary values, we have to find the maximal set of states A_{f-1} that satisfies:

$$\text{Max}\{ \sum_{x_i \in L_f} y_i + \sum_{x_i \in A_{f-1}} y_i \} = \text{Max} \sum_{y} \sum_{x_i \in L_f} y_i = k(L_f) . \quad (3.36)$$

Moreover, the enlarged Hamiltonian at t_f^- must lie on Y in such a way that all operating points in Y_{f-1} satisfy equation (3.36). But as said before, all costates corresponding to states in L_f have the same value at t_f^-, so that the enlarged Hamiltonian will contain the $(I_{f-1} \cup A_{f-1})$-face (3.36). Hence Theorem 3.1 implies that

(i) $\lambda_i(t_f^-) = 0$ for all $x_i \in B_{f-1}/A_{f-1}$,

(ii) the values of all $\lambda_i(t_f^-)$ are identical for all i such that $x_i \in L_f \cup A_{f-1}$,

where (i) follows from Assertion 3.3, and (ii) follows from Assertion 3.4 and the fact that all costates corresponding to states in L_f have the same value. Now, clearly there exists at least a point on the $(I_{f-1} \cup A_{f-1})$-face, satisfying

$$\left. \begin{array}{l} y_i > 0, \quad \forall x_i \in L_f = I_{f-1} \\ \\ y_i = 0, \quad \forall x_i \in B_{f-1} \end{array} \right\} . \quad (3.37)$$

Moreover, since we require that all costates corresponding to states travelling on boundary arcs in the interval (t_f, t_{f-1}) be at their leave-the-boundary

values, (i) and (ii) hold for all $\tau \epsilon (t_f, t_{f-1})$, so that the point (3.37) is an optimal solution for all $\tau \epsilon (t_f, t_{f-1})$. Therefore we have:

$$\dot{\lambda}_i(\tau) = -1, \quad \forall x_i \epsilon L_f \cup A_{f-1} = I_{f-1} \cup A_{f-1}$$

$$\dot{\lambda}_i(\tau) = \lambda_i(\tau) = 0, \quad \forall x_i \epsilon B_{f-1}/A_{f-1}$$

(3.38)

$$\forall \tau \epsilon (t_f, t_{f-1})$$

Now we perform the induction step: Consider the time interval (t_{p+1}, t_p), and assume that the set of operating points Y_p, contains at least one point satisfying:

$$y_i \geq 0, \quad \forall x_i \epsilon I_p$$

$$y_i = 0, \quad \forall x_i \epsilon B_p \quad .$$

(3.39)

We denote by A_p the maximal set of states satisfying

$$\underset{Y}{Max} \{ \sum_{x_i \epsilon I_p} y_i + \sum_{x_i \epsilon A_p} y_i \} = k(I_p) \quad .$$

Keeping the costates corresponding to states in B_p at their leave-the-boundary values at every instant on the interval will result by Theorem 3.1 in:

$$\lambda_i(\tau) = 0, \quad \forall x_i \epsilon B_p/A_p \quad ,$$

(3.40a)

$$\lambda_i(\tau) = \lambda_j(\tau) \quad \forall x_i \epsilon A_p, \quad x_j \epsilon I_p \quad .$$

(3.40b)

Moreover, from the existence of a point satisfying (3.39) during the entire interval follows that:

$$\dot{\lambda}_i(\tau) = -1, \quad \forall x_i \epsilon I_p$$

$$\forall \tau \epsilon (t_p, t_{p+1}) \quad ,$$

(3.41)

therefore from (3.40) and (3.41) we have:

$$\dot{\lambda}_i(\tau) = \lambda_i(\tau) = 0, \quad \forall x_i \in B_p/A_p \quad , \tag{3.42a}$$

$$\dot{\lambda}_i(\tau) = -1; \quad \lambda_i(\tau) > 0, \quad \forall x_i \in I_p \cup A_p \quad , \tag{3.42b}$$

$$\forall \tau \in (t_p, t_{p+1}) \quad ,$$

where $\lambda_i(\tau) > 0$ follows from Assertion 3.1 of Theorem 3.1.

Now consider the boundary junction time t_p where we allow the set of states L_p to leave the boundary, so that $I_{p-1} = I_p \cup L_p$. Noting that we have kept all costates corresponding to states in B_p at their leave-the-boundary values during all the interval (t_p, t_{p+1}), we realize that at t_p (3.40) is satisfied, namely at the junction time the Hamiltonian does not change its orientation. Therefore in order to reach points in Y_{p-1} (from a point in Y_p), we have to perform pivots on the subnetwork corresponding to states in $I_p \cup A_p$ (see Figure 3.6), increasing in this way the flows on the links corresponding to y_i, where $x_i \in L_p \cap A_p$, and then to increase the flows on the links corresponding to y_i, where $x_i \in L_p \cap (B_p/A_p)$.
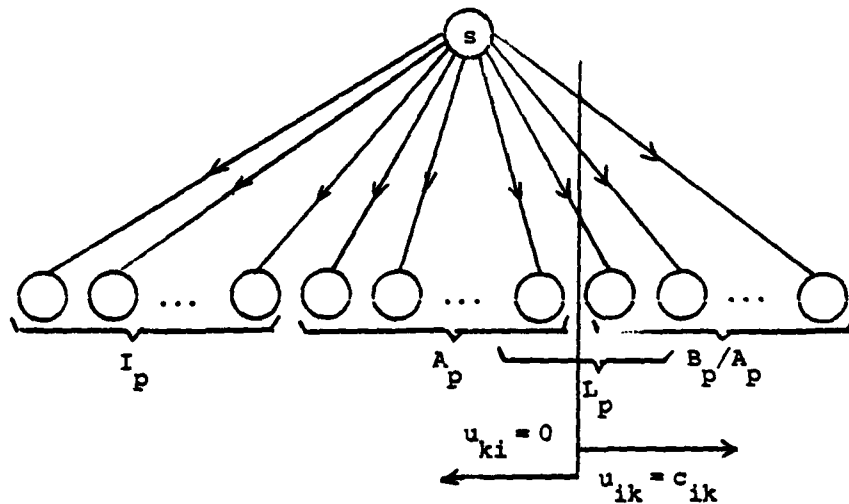
Figure 3.6 - Network representing transition between Y_p to Y_{p-1}.

As a result of the increase of flow on a link y_i corresponding to an $x_i \in L_p \cap A_p$, there is a decrease of flow on some link y_j corresponding to an $x_j \in I_p$, however, it is clear that we can always increase the flow on y_i in such a small amount that y_j will still remain nonnegative for all $x_j \in I_p$. Moreover, the increase of flow on links y_i corresponding to $x_i \in (B_p/A_p) \cap L_p$ does not result in decreases of flows on other slack links so that Y_{p-1} will contain points satisfying

$$y_i \geq 0 \quad \forall x_i \in I_{p-1} \quad ,$$
$$y_i = 0 \quad \forall x_i \in B_{p-1} \quad . \tag{3.43}$$

Now $I_p \subset I_{p-1}$ so that when maximal flow is achieved on the network corresponding to I_{p-1} (i.e., network with slack links y_i for all $x_i \in I_{p-1}$ only), all minimal cuts of I_p will be filled. In addition, the sets I_p and $I_p \cup A$ have the same minimal cuts, therefore:

$$A_p \subset A_{p-1} \cup L_p \quad , \tag{3.44}$$

where A_{p-1} is the maximal set of states satisfying

$$\underset{y}{\text{Max}}\{ \sum_{x_i \in I_{p-1}} y_i + \sum_{x_i \in A_{p-1}} y_i \} = k(I_{p-1}) \quad .$$

Now, from the existence of a point satisfying (3.43), and from the same considerations as in the interval (t_p, t_{p+1}), we have

$$\dot{\lambda}_i(\tau) = \lambda_i(\tau) = 0 \qquad \forall x_i \in B_{p-1}/A_{p-1}$$
$$\dot{\lambda}_i(\tau) = -1; \quad \lambda_i(\tau) > 0; \quad \forall x_i \in I_{p-1} \cup A_{p-1} \tag{3.45}$$
$$\forall \tau \in (t_{p-1}, t_p) \quad .$$

Moreover, since $I_p \subset I_{p-1}$, from (3.45) follows that

$$\dot{\lambda}_i(\tau) = -1; \quad \lambda_i(\tau) > 0; \quad \forall x_i \in I_p \quad ,$$
$$\forall \tau \in (t_{p-1}, t_p) \quad , \tag{3.46}$$

and since $L_p \subset I_{p-1}$, from (3.44) and (3.45) follows that:

$$\dot{\lambda}_i(\tau) = -1; \quad \dot{\lambda}_i(\tau) > 0, \quad \forall x_i \in A_p,$$

$$\forall \tau \in (t_{p-1}, t_p) \quad . \tag{3.47}$$

Therefore, equations (3.39), (3.42), (3.43), (3.46) and (3.47) provide the induction step that proves the theorem.

$$\square \text{ Theorem 3.2.}$$

The costate trajectories (3.35) satisfy the necessary conditions (which are also sufficient) derived in [M2]. Moreover, these trajectories were obtained by checking at every moment the global Hamiltonian, thus insuring that values of the costates corresponding to states on the boundary can always be found such that the solutions obtained by considering only the restricted Hamiltonian are also globally optimizing solutions. The above arguments yield to the following corollary:

Corollary 3.2

Any solution to the constrained optimization problem (see [M2], p. 64) is also a globally optimizing solution.

$$\square \text{ Corollary 3.2.}$$

Theorem 3.3

The set of optimal controls does not switch between boundary junction times; that is, there are no break points between boundary junctions.

Proof

Consider the time interval (t_p, t_{p+1}) and denote:

$$\underset{Y}{\text{Max}} \sum_{x_i \in I_p} \lambda_i(\tau) y_i = a(\tau) \quad .$$

Now consider the time $\tau_1 \in (t_p, t_{p+1})$. The set of operating points is given by

$$Y_p(\tau_1) = \{\underline{y} \in Y / \sum_{x_i \in I_p} \lambda_i(\tau_1) y_i = a(\tau_1)\} \cap \mathbf{R}^{\sigma_p} , \qquad (3.48)$$

where the basis vectors of $\mathbf{R}^{\sigma_p}$ are the elements of I_p.

Recall that all points in (3.48) satisfy

$$\sum_{x_i \in I_p} y_i = k(I_p) . \qquad (3.49)$$

Now consider the time $\tau_2 \in (t_p, t_{p+1})$, where $\tau_2 = \tau_1 + \Delta\tau$. The set of operating points is given by

$$Y_p(\tau_2) = \{y \in Y / \sum_{x_i \in I_p} \lambda_i(\tau_2) y_i = a(\tau_2)\} \cap \mathbf{R}^{\sigma_p} . \qquad (3.50)$$

But every point in (3.50) satisfies (3.49), and also by equation (3.42b) we have

$$\dot{\lambda}_i(\tau) = -1, \ \forall x_i \in I_p$$

$$\forall \tau \in (t_p, (t_{p+1}) .$$

Therefore:

$$\underset{y}{\text{Max}} \sum_{x_i \in I_p} \lambda_i(\tau_2) y_i = \underset{y}{\text{Max}}\{ \sum_{x_i \in I_p} \lambda_i(\tau_1) y_i + \Delta\tau \sum_{x_i \in I_p} y_i \} = \underset{y}{\text{Max}} \sum_{x_i \in I_p} \lambda_i(\tau_1) y_i + \Delta\tau \, k(I_p) .$$

Thus:

$$Y_p(\tau_1) = Y_p(\tau_2) ,$$

so that the set of optimal controls does not switch between boundary junction times.

$\square$ Theorem 3.3.

Theorem 3.4

There is one subregion per region with respect to any set of state variables leaving the boundary.

Proof

Consider the feedback control region R_p constructed from the set of

optimal solutions Y_p on $(-\infty, t_{p+1})$. Also, let L_p be the set of state variables which we choose to allow to leave from R_p backwards in time, where $L_p \subset B_p$. According to the definition of subregions (see [M2], p. 57), we must show that the state variables in L_p leave with the same set of optimal controls from every point of R_p. Recall that by Theorem 3.3 the optimal trajectories associated with the state variables in L_p, leaving R_p, do not experience a break as time runs to minus infinity, so that to prove that there is one sub-region per region is equivalent to showing that the set of optimal controls associated with the state variables in L_p leaving the boundary is the same for every boundary junction time $t_p \in (-\infty, t_{p+1})$. Note that we need not be concerned with the common breakwall sequences (see [M2], p. 56), since by Theorem 3.3 there are no breakwalls. Now consider two different boundary junction times, t_{p_1}, $t_{p_2} \in (-\infty, t_{p+1})$ and assume that $t_{p_1} > t_{p_2}$. By Theorem 3.3 we have that

$$Y_{p-1}(t_{p_1}) = \{\underline{y} \in \mathcal{Y} / \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(t_{p_1})y_i = a(t_{p_1})\} \cap \mathbb{R}^{\sigma_p + \rho_p} \; , \qquad (3.51)$$

and

$$Y_{p-1}(t_{p_2}) = \{\underline{y} \in \mathcal{Y} / \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(t_{p_2})y_i = a(t_{p_2})\} \cap \mathbb{R}^{\sigma_p + \rho_p} \qquad (3.52)$$

where

$$a(\tau) = \underset{\mathcal{Y}}{\text{Max}} \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(\tau)y_i \; .$$

The coordinates of $\mathbb{R}^{\sigma_p + \rho_p}$ are the elements of I_{p-1}, and A_{p-1} is defined as in Theorem 3.2.

Now, by Theorems 3.2 and 3.3 we know that

$$\dot{\lambda}_i(\tau) = -1, \; \forall x_i \in I_p \cup A_p$$

$$\lambda_i(\tau) = 0, \; \forall x_i \in B_p / A_p$$

$$\forall \tau \in (-\infty, t_{p+1})$$

where A_p is defined as in Theorem 3.2.

Hence:

$$\lambda_i(t_{p_2}) = \lambda_i(t_{p_1}) = 0, \quad \forall x_i \in (I_{p-1} \cup A_{p-1})/(I_p \cup A_p) \quad ,$$

$$\lambda_i(t_{p_2}) = \lambda_i(t_{p_1}) + (t_{p_1} - t_{p_2}), \quad \forall x_i \in I_p \cup A_p \quad ,$$

therefore

$$\text{Max}_y \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(t_{p_2})y_i = \text{Max}_y \{ \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(t_{p_1})y_i + (t_{p_1} - t_{p_2}) \sum_{I_{p-1} \cup A_{p-1}} y_i \} =$$

$$= \text{Max}_y \sum_{I_{p-1} \cup A_{p-1}} \lambda_i(t_{p_1})y_i + (t_{p_1} - t_{p_2})k(I_{p-1}) \quad ,$$

so that

$$Y_{p-1}(t_{p_1}) = Y_{p-1}(t_{p_2}) \quad ,$$

then the set of optimal controls associated with state variables in L_p leaving the boundary from R_p, is the same for every $t_p \in (-\infty, t_{p+1})$.

□ Theorem 3.4.

Section 4

THE ALGORITHM

This section is devoted to the description of a simplified algorithm
for building the feedback solution to the minimum delay dynamic message rout-
ing problem for single destination networks with all unity weightings in the
cost functional. The simplifications obtained arise from the special proper-
ties of these kind of networks, as discussed in Section 3, and from the new
approach to solve the linear programs presented in Section 2.

In Part A we state the algorithm while explaining its steps. In
Part B we present a method for obtaining all optimal solutions of the linear
programming problems required by the algorithm and finally Part C brings an
example that provides a good insight of the performance of the algorithm.

A. Statement of the Algorithm

Operation 1

List all possible trajectories in the state space, by writing all pos-
sible sequences of states leaving the boundary backwards in time.

□ Operation 1.

Recall that every optimal trajectory in the state space is character-
ized by a control switch time sequence. Each switch time represents a transi-
tion between two neighboring feedback control regions. Now by Theorems
3.2 - 3.4 there are no returns of states to the boundary, there are no breaks
in the optimal controls between junction times and there is only one subregion
per region. Therefore every trajectory is characterized only by departures
of state variables from the boundary, the boundary junction times might be
arbitrarily taken and between two successive boundary junction times the set

of optimal controls does not change. Thus, Operation 1 takes into account all possible trajectories in the state space, so that the construction of feedback control regions based on these sequences insures a complete covering of the state space. Such a typical sequence looks as follows:

$$\{L_f\}, \{L_{f-1}\}, \ldots, \{L_{p-1}\}, \{L_p\}, \{L_{p-1}\}, \ldots, \{L_{f-n+1}\}$$

The number of possible sequences $Q(n)$ is given by (see [E1], p. 68).

$$Q(n) = \sum_{r=1}^{n} \sum_{i=0}^{r} (-1)^i \binom{r}{i} (r-i)^n$$

Notice that the complexity of Operation 1 is of the order of n^n, so that clearly the algorithm is implementable only for "small" networks.

Example 4.1

Consider a network with three state variables x_1, x_2 and x_3. The sequences are:

	t_f	t_{f-1}	t_{f-2}
1.	x_1	x_2	x_3
2.	x_1	x_3	x_2
3.	x_1	(x_2, x_3)	-
4.	x_2	x_1	x_3
5.	x_2	x_3	x_1
6.	x_2	(x_1, x_3)	-
7.	x_3	x_1	x_2
8.	x_3	x_2	x_1
9.	x_3	(x_1, x_2)	-
10.	(x_1, x_2)	x_3	-
11.	(x_1, x_3)	x_2	-
12.	(x_2, x_3)	x_1	-
13	(x_1, x_2, x_3)	-	-

□ Example 4.1

Operation 2

Derive the networks corresponding to every possible combination of state variables leaving the boundary. Apply the algorithm of Maximal Flow to the networks and find the corresponding flows.

$\square$ Operation 2.

From Operation 2 we obtain the first optimal solutions to the linear programs (as explained in Parts B and C of Section 2). Also, the operation provides the essential tools for obtaining the simplifications of the algorithm as will be shown in Operation 3.

The number of networks for which the Maximal Flow Algorithm is to be applied is $2^n - 1$. The complexity of this operation is therefore exponential in the number of nodes.

In Appendix B we provide a computer program in Fortran for finding the maximal flow in these networks, based on the algorithm of Edmons and Karp (see [H1]).

Example 4.2

All the possible combinations of states leaving the boundary in example 4.1 are:

$$\{x_1\},\{x_2\},\{x_3\},\{x_1,x_2\},\{x_1,x_3\},\{x_2\ x_3\},\{x_1,x_2,x_3\} \quad .$$

$\square$ Example 4.2

Operation 3

For every sequence of state variables leaving the boundary listed in Operation 1, and considering the results obtained in Operation 2, execute the following steps:

(a) Consider every set L_p of the sequence and check if there is a sub-
   set of states B such that $B \subset L_p$, and such that there is a set B'
   where $I_p \cup B \subset B'$, and such that the same value of maximal flow
   corresponds to both sets, $I_p \cup B$ and B'.

(b) If (a) is satisfied, check if in a set L_q of the same sequence,
   where $q < p$ there is at least one state variable x_j such that
   $x_j \in B'/(I_p \cup B)$.

(c) If (b) is satisfied, erase from the list of Operation 1 the sequence
   being currently checked.

$\square$ Operation 3.

In order to justify Operation 3, consider first the following notation:

$$\{I_{f-1}\},\{I_{f-2}\},\ldots,\{I_p\}, \{I_{p-1}(B)\},\ldots \quad . \tag{4.1}$$

The sequence (4.1) represents all trajectories in the state space character-
ized by the sets of states travelling on interior arcs in every time interval.
In (4.1), the notation $\{I_{p-1}(B)\}$ means that the set of states B is such
that $B \subset L_p$, that is, the states in B leave the boundary at t_p. Notice
that $B \subset I_{p-1}$.

Now consider the following sequence satisfying (a) and (b):

$$\{I_{f-1}\},\ldots,\{I_p\}, \{I_{p-1}(B)\},\ldots,\{I_q\},\{(I_{q-1}(x_j)\},\{I_{q-2}\},\ldots \tag{4.2}$$

where $x_j \in B'/(I_p \cup B)$.

Now consider the following sequence:

$$\{I_{f-1}\},\ldots,\{I_p\}, \{I_{p-1}(B)\},\ldots,\{I_q \cup x_j\},\{I_{q-1}\},\{I_{q-2}\},\ldots \quad . \tag{4.3}$$

The only difference between sequences (4.2) and (4.3) is that in (4.2),
x_j leave the boundary at t_q while in (4.3) x_j leave the boundary at t_{q+1}.
Notice that the existence of a sequence as (4.3) is insured since Operation 1

requires the listing of every possible sequence.

We claim that all feedback control regions constructed from sequence (4.2) are also constructed from sequence (4.3) so that sequence (4.2) is redundant. The reasons for that appear in the following arguments:

By the considerations of Part D in Section 2, we know that in order to find all optimal solutions (to construct the corresponding feedback control region) corresponding to the time interval (t_{g+1}, t_g), we find all optimal solutions corresponding to the states in I_g leaving the boundary, and among these solutions we choose those that maximize the expression

$$\sum_{\substack{x_i \in I_g}} \lambda_i(t_g) y_i \ .$$

Now it is not difficult to see, that by Theorem 3.2 the costate values in both sequences (4.1) and (4.2) are the same at all times. Moreover, note that if all optimal solutions corresponding to the states in $I_p \cup B$ leaving the boundary are also optimal solutions to the problem corresponding to the states in B' leaving the boundary, then all optimal solutions corresponding to the states in $I_p \cup B \cup D$ leaving the boundary (for some set of states D) are also optimal solutions of the problem corresponding to the states in $I_p \cup B \cup D \cup E$ leaving the boundary, where $E \subset B'/I_p \cup B$. The above is easily seen by noting that all minimal cuts corresponding to $I_p \cup B$ are also minimal cuts of the network corresponding to $I_p \cup B \cup E$, therefore all minimal cuts corresponding to $I_p \cup B \cup D$ are also minimal cuts of the network corresponding to $I_p \cup B \cup D \cup E$.

Now returning to sequences (4.2) and (4.3), we have that

$$I_q^2 = I_q^1 \cup x_j \ , \tag{4.4}$$

(where I_q^1 and I_q^2 denote the sets of states travelling on interior arcs in the time interval (t_{q+1}, t_q) in sequences (4.2) and (4.3) respectively).

Moreover we can write

$$I_q^1 = (I_q^1/I_p \cup B) \cup I_p \cup B \quad , \tag{4.5}$$

so that from (4.4) and (4.5) we have

$$I_q^2 = (I_q^1/I_p \cup B) \cup I_p \cup B \cup x_j \quad . \tag{4.6}$$

But from the above reasoning all optimal solutions corresponding to (4.5) are also optimal solutions of the problem corresponding to (4.6), moreover:

$$I_g^2 = I_g^1 \ , \ \forall \ g \leq q \quad .$$

Therefore the sequence (4.2) is redundant.

Example 4.3

If in example 4.2 we obtain the same maximal flow value in the cases corresponding to the sets of states x_1 and $\{x_1, x_2, x_3\}$ leaving the boundary, then we erase from the list of example 4.1 the sequences numbered (1), (2), (3), (4), (7), (10) and (11).

$\square$ Example 4..3.

Operation 4

For every state variable combination L_f leaving the boundary at t_f for which there is not a combination L_f' such that $L_f \subset L_f'$, and the maximal flow values corresponding to both combinations are equal, find all the optimal basic solutions of the corresponding linear programming problem starting with an optimal (extended) basic solution as explained in Parts B and C of Section 2, and using the algorithm provided in Part B of this section.

$\square$ Operation 4.

The results obtained from Operation 4 are the basic data needed for con- structing feedback control regions (the construction is carried out in Opera- tion 5).

Operation 5

For every remaining sequence (after execution of Operation 3) from the list of Operation 1, carry out the following steps for every boundary junction time t_p, in a sequential order, starting at t_f:

(a) Set

$$\dot{\lambda}_i(\tau) = -1, \ \forall \ x_i \in I_{p-1}$$

$$\forall \ \tau \in (-\infty, t_p)$$

$$\lambda_i(\tau) = 0, \ \forall x_i \in B_{p-1}$$

$$\forall \ \tau \in (t_{p-1}, t_p)$$

with

$$\lambda_i(t_f) = 0, \ \forall \ \{x_i / i \in N\} \ .$$

(b) Set arbitrarily $t_p - t_{p-1} = 1$.

(c) From among all solutions $\{y^*\}$ obtained in Operation 4 with $L_f = I_{p-1}$ choose those that maximize the expression:

$$\sum_{x_i \in I_{p-1}} \lambda_i(t_{p-1}) y_i \ .$$

(d) Transform the solutions obtained in (c) to the set of rays V_{p-1} (see [1], p. 58) and construct the convex polyhedral cone:

$$R_{p-1} = C_o(R_p \cup V_{p-1})/R_\rho \ ,$$

where $C_o(\cdot)$ denotes the convex hull.

(e) Consider all sequences that are identical to the current sequence until the current time interval (t_{p-1}, t_p). The costate values of such sequences in interval (t_{p-1}, t_p) will be identical to the costate values corresponding to the considered sequence, and the corresponding feedback control region will be built only once. This

insures that we avoid multiple construction of the same control region.

□ Operation 5.

Step (a) follows from the costate trajectories depicted in Theorem 3.2. Notice that in the sequences left after Operation 3, the costate λ_i starts to change only from the boundary junction time in which its corresponding state leaves the boundary, so that the calculation of the costate values is an easy task. Step (b) follows from the fact that there is only one subregion per region with respect to any set of state variables leaving the boundary (as proved in Theorem 3.4), so that we may arbitrarily choose the boundary junction times. Step (c) follows from Part D of Section 2. Step (d) follows from the Constructive Dynamic Programming Algorithm stated in [M2]. Finally, step (e) is a simple conclusion that saves superfluous computational work.

B. The Method for Finding all Solutions
   to the Linear Programming Problems

One of the major problems in the implementation of the algorithm described in Part A of this section arises from the need of finding all solutions to every linear program required in Operation 4 of the algorithm. In geometrical terms, the problem is to find all the extremal points of the convex polyhedron formed by the intersection between the enlarged Hamiltonian and the γ-constraint figure lying in the positive orthant of the γ-space. Fortunately, the above linear programs are in general highly degenerate so that as it was clarified in Part C of Section 2, the number of linear programs we actually have to solve is greatly reduced.

The problem of finding all solutions to linear programs has been widely investigated. See for example [B1], [Ch1], [Ch2], [M4], [R1]. In [B2], we find a report on computational experience gained using the algorithms

presented in [Ch1] and [Ch2], as applied to optimal routing problems. From this report it is seen that even for very small size networks the amount of computation and memory required is excessive, and also the numerical sensitivity of the algorithm is extremely high.

We proceed now with the description of the method to find all the solutions of the linear programs that we propose and later we shall discuss its advantages. The algorithm is based on the method of pivoting on networks. For a review of this method the reader is referred to Appendix A.

Starting at an optimal extended basic solution, the algorithm picks a nonbasic link and looks for the cycle formed by basic links and the considered nonbasic link. There always exists such a (unique) cycle since every extended basic solution corresponds to a spanning tree in the network. In order to find the cycle the algorithm utilizes a labeling technique that will be described later on. After finding the cycle we perform a pivot on it, reaching in this way either a new extremal solution or, if the cycle is degenerate, another representation of the original extremal solution. If the degeneracy is of a high degree, we may find with a single pivot several different representations of the extremal solution. Every new solution or representation obtained is numbered and kept in memory if and only if it does not exist there already. The above process is performed for all nonbasic links corresponding to every solution and/or representation kept in memory until it is not possible to reach a new solution or representation. In this way we insure that at the end of the process we had reached all optimal solutions. Since we are not concerned with different representations of extremal solutions when constructing feedback control regions at the end of the process, we can delete these representations retaining only one representation per extremal solution. In order to start the process, we reach an initial extended basic optimal solution as depicted in Parts B and C of Section 2.

Now, to understand the need of searching also the different representations of every extremal solution, let us consider the following example.

Example 4.4

Given the network depicted in Figure 4.1, the problem is to find all optimal solutions corresponding to all states leaving the boundary.
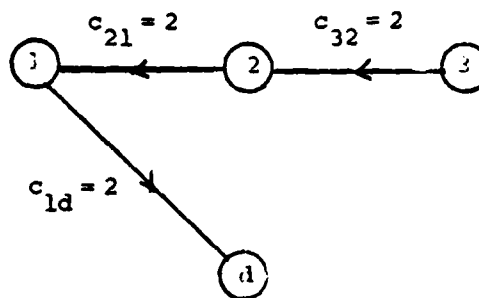
Figure 4.1 - Network of Example 4.4

After finding the minimal cut of the network (recall that to find a first optimal basic solution, we add a new node (the source) and links with no capacity constraints connecting the source with every node of the network, and we apply the maximal flow algorithm), we choose the first basic optimal solution shown in Figure 4.2-a. From this solution and by pivoting on the cycle corresponding to the nonbasic link (3,2), we reach another extremal solution as depicted in Figure 4.2-b. Note that from the first solution, pivoting on the cycle corresponding to the nonbasic link (s,2) does not lead to a new solution but just to another representation of the first optimal solution. Clearly, from the representation of the extremal solution of Figure 4.2-b we cannot reach a new extremal solution. Only starting at the representation shown in Figure 4.2-c can we reach the new extremal solution depicted in Figure 4.2-d.

(a)

(b)

- - - - nonbasic link

——— basic link

(c)

(d)
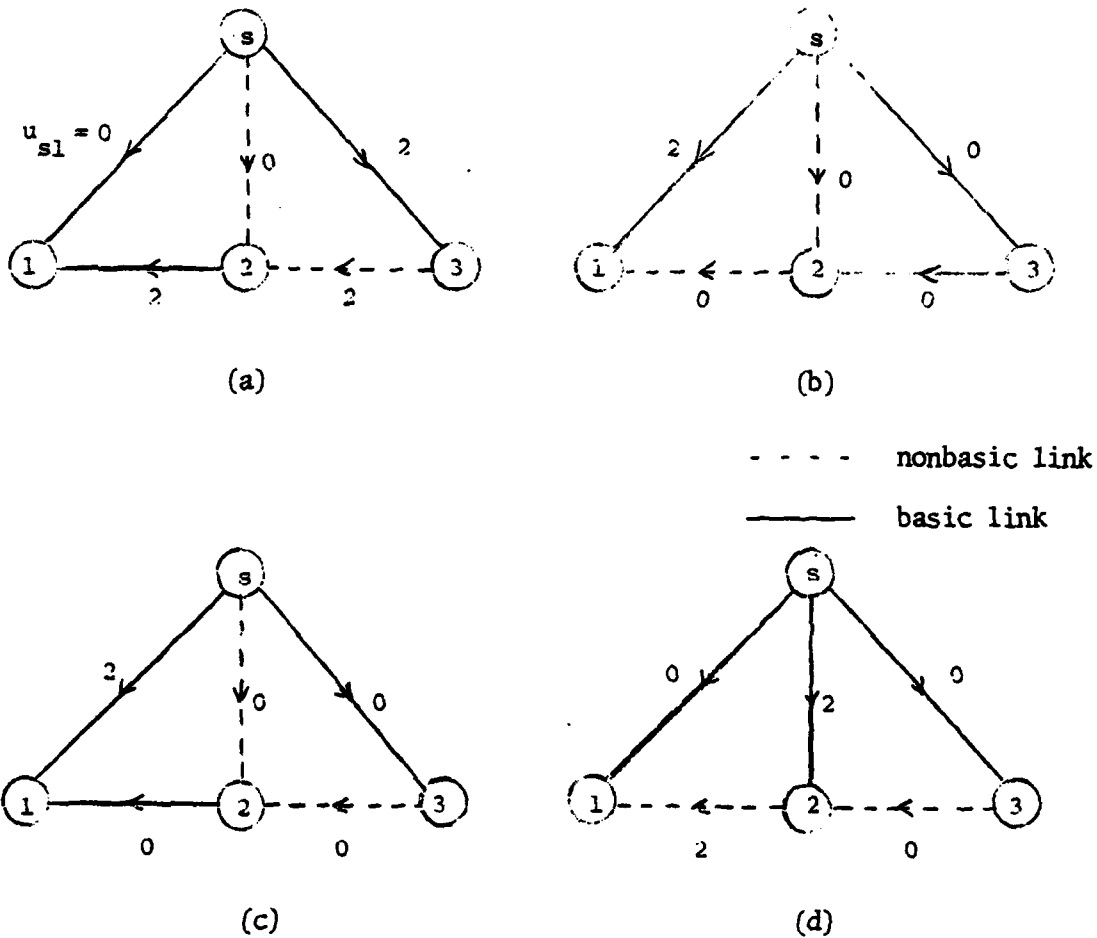
Figure 4.2 - Obtaining extremal points in the network of Example 4.4.

□ Example 4.4.

Now we describe the labeling technique used by the algorithm to find pivot cycles. First we assign a number to every node and link in the network. For example see Figure 4.3:
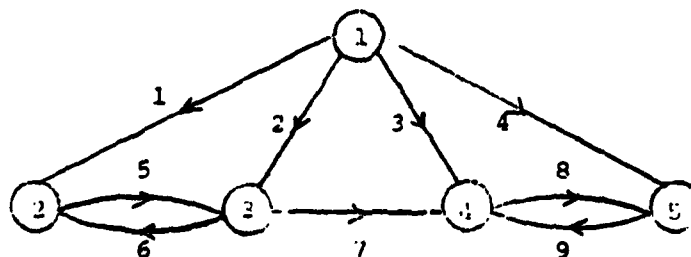
Figure 4.3 - Numbering nodes and links.

Second, we define the following arrays:

C(I) = Capacity of the link I.

F(I,J) = Flow on the link I at the J-th solution.

BASE(I,J) = Type of the link I (basic or nonbasic) at the J-th solution.

LOUT(I) = Exit node of link I.

LIN(I) = Entrance node of link I.

where:

$$BASE(I,J) = \begin{cases} 1 & \text{if the link I is basic} \\ 0 & \text{if the link I is nonbasic .} \end{cases}$$

Now if link K is nonbasic, in order to find the pivot cycle we search the arrays LOUT(I) and LIN(I) and we find all the incident links to node LIN(K) (where LIN(K) is the entrance node of the link K). After finding all these links we consider only those that are basic, and for every one of them we carry out the same procedure until we obtain a basic link entering (exiting) into (from) node LOUT(K). To every link searched in the above procedure, the number of the preceding link of the cycle is matched so that at the end of the procedure a well-defined cycle is obtained. After finding the cycle, and by checking the arrays C(I) and F(I,J) we obtain the maximal change of flow on the cycle and we perform the pivot. As said before, when the cycle is degenerate, we reach all possible different representations by performing "dummy" pivots, that is declaring the nonbasic link as basic, and the basic link that does not allow to change the flow as nonbasic.

Standard labeling techniques assign numbers only to the nodes of the network so that the links are identified by a two-dimensional array (I,J), where I is the exit node of the link and J its entrance node. However,

note that in order to use this technique we must define in our case two arrays that are three-dimensional, the first accounting for the flows on the links, the second for the type of links. Moreover a two-dimensional array for the link capacities is needed, thus making use of an excessive and wasteful amount of memory, especially when the number of solutions and representations is large. Clearly, by defining only two-dimensional arrays (as we do) we save a large amount of memory. Notice also that, when finding all the solutions to the linear programs by means of pivoting on networks, every solution or representation is identified by a solution vector and a link-type vector instead of a tableau (as is done when utilizing simplex methods) saving again a great deal of memory.

The main problem of the method is that we do not know a priori the number of solutions and representations of the linear program, so that we do not know the dimension needed for the arrays that will contain the solutions or representations. Truly, we can find an upper bound to the number of solutions and representations since every basic solution corresponds to a spanning tree and it is possible to calculate the number of spanning trees in the network, however, not every spanning tree corresponds to a feasible basic solution. For example, in the network depicted in Figure 4.4 a non-feasible spanning tree is shown.
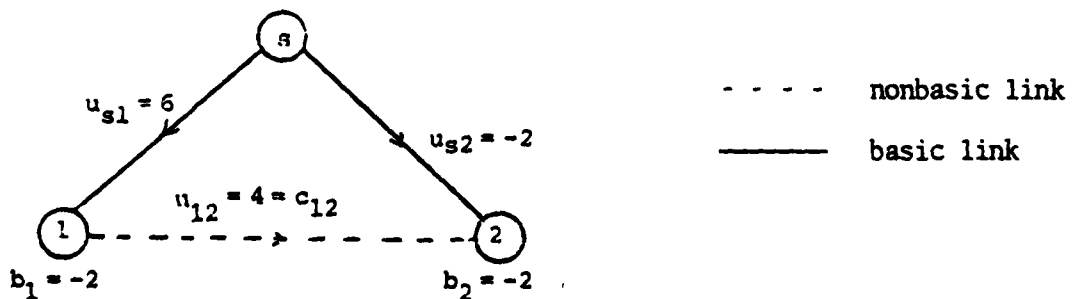
Figure 4.4 - Example of a non-feasible spanning tree.

Therefore, in general, the number of feasible solutions is less than the number of spanning trees, and for practical purposes the dimension of the arrays having a coordinate defining the number of solutions and representations is estimated and fixed accordingly.

We present now several examples that will test the efficiency of the method. All examples apply to the case when all the states leave the boundary (that is, the case in which we obtain the maximal number of solutions) and they were solved by means of a Fortran computer program (see Appendix B) based on the method described above. All programs were run on an IBM 370/168 computer, and for every example the maximal memory region provided was 512K.

In Figure 4.5 we show the tested networks with the corresponding numbers of optimal solutions and representations, and the CPU time consumed.
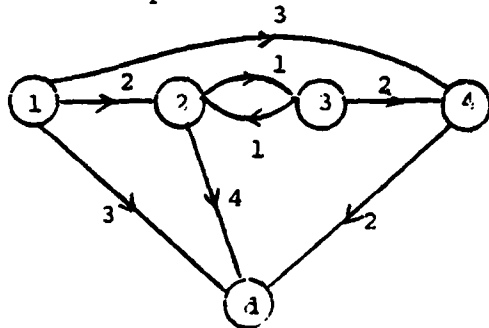
Number of solutions: 24

Number of solutions
and representations: 48

CPU time: 1.78 sec.

(a)
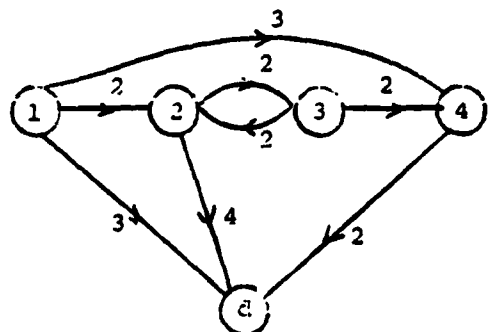
Number of solutions: 20

Number of solutions
and representations: 58

CPU time: 1.59 sec.

(b)
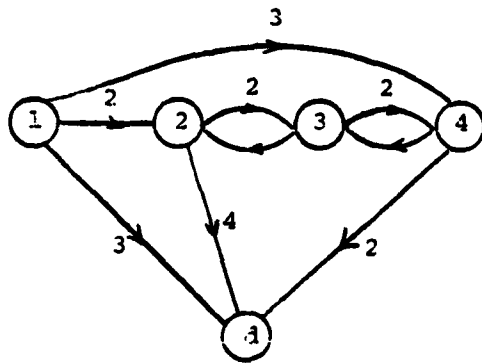
(c)

Number of solutions: 60

Number of solutions
and representations: 156

CPU time: 5.41 sec.

(d)
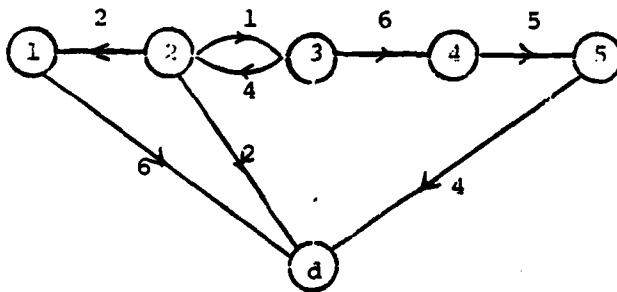
Number of solutions: 34

Number of solutions
and representations: 78

CPU time: 2.2 sec.

(e)
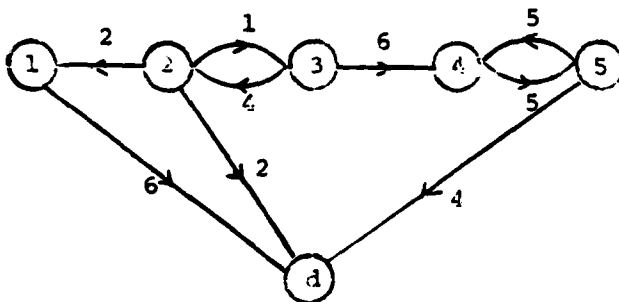
Number of solutions: 68

Number of solutions
and representations: 184

CPU time 7.41 sec.

Figure 4.5 - Example of finding all optimal solutions.

From the examples of Figure 4.5 we obtain an insight into the change in the number of feasible spanning trees when changing capacities of some links (see 4.5a and 4.5b) and the great increase in the number of feasible spanning trees when adding a single link to the network (see 4.5b and 4.5c, 4.5d and 4.5e).

Note that since the capacities are integers and the only arithmetical operations in the method are additions and substractions, no numerical sensitivity problems arise.

Now we anal.ze briefly the complexity of the method for finding all optimal solutions. Consider first the non-degenerate case, that is, the case in which every optimal solution satisfies:

$$y_i > 0, \quad i = 1,2,\ldots,n , \tag{4.7}$$

where n is the number of nodes in the network. Denote also by ℓ the number of links in the network. Clearly, if (4.7) is satisfied at every optimal solution, the links corresponding to the slack variables "y" are basic, and the remaining links are non-basic. Since the flow on a non-basic link (i,j) must be only zero or c_{ij}, the number of optimal (basic) solutions is $2^{(\ell-n)}$. This explains the great increase in the number of solutions when adding a single link to the network. In the degenerate case we can only estimate an upper bound for the number of solutions and representations. This upper bound is given by multiplying the number of spanning trees of the network by $2^{(\ell-n)}$. The conclusion arising from the above arguments is that owing to its exponential complexity the method can actually be implemented only for relatively small networks.

C. Example of Feedback Solution to the Dynamic Routing Problem of a
   Single Destination Network with all Unity Weightings in the Cost Functional

In order to illustrate the algorithm described in Part A of this section we present here an example of a four-state variable network for which we execute some of the operations of the algorithm. The network is depicted in Figure 4.6.
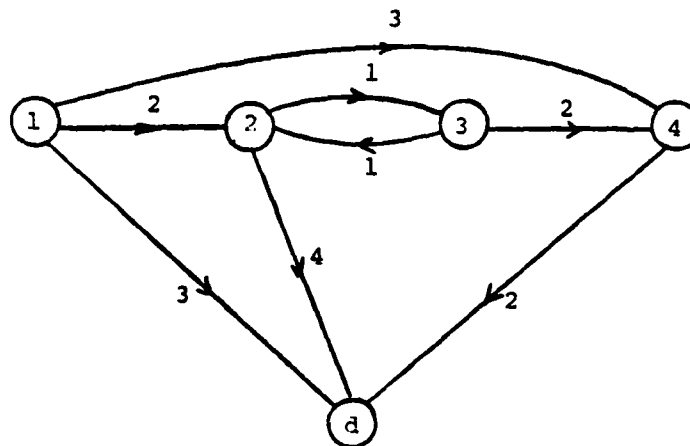
Figure 4.6 - Network to illustrate the algorithm.

Operation 1

In this operation we list all possible sequences of states leaving the boundary. Since there are four states, we obtain $Q(4) = 75$ different sequences.

t_{f-3}	t_{f-2}	t_{f-1}	t_f
x_4	x_3	x_2	x_1
x_3	x_4	x_2	x_1
x_2	x_4	x_3	x_1
x_4	x_2	x_3	x_1
x_2	x_3	x_4	x_1
x_3	x_2	x_4	x_1
x_4	x_3	x_1	x_2
x_3	x_4	x_1	x_2
x_4	x_1	x_3	x_2
x_1	x_4	x_3	x_2

	x_1	x_3	x_4	x_2
	x_3	x_1	x_4	x_2
	x_4	x_2	x_1	x_3
	x_2	x_4	x_1	x_3
	x_4	x_1	x_2	x_3
	x_1	x_4	x_2	x_3
	x_2	x_1	x_4	x_3
	x_1	x_2	x_4	x_3
	x_3	x_2	x_1	x_4
1	x_2	x_3	x_1	x_4
	x_3	x_1	x_2	x_4
	x_1	x_3	x_2	x_4
2	x_2	x_1	x_3	x_4
3	x_1	x_2	x_3	x_4
		x_4	x_2,x_3	x_1
		x_3	x_2,x_4	x_1
		x_2	x_3,x_4	x_1
4		x_1	x_3,x_4	x_2
		x_4	x_1,x_3	x_2
		x_3	x_1,x_4	x_2
		x_4	x_1,x_2	x_3
		x_2	x_1,x_4	x_3

	x_1	x_2, x_4	x_3
	x_3	x_1, x_2	x_4
5	x_1	x_2, x_3	x_4
6	x_2	x_1, x_3	x_4
	x_3, x_4	x_2	x_1
	x_2, x_4	x_3	x_1
	x_2, x_3	x_4	x_1
	x_3, x_4	x_1	x_2
	x_1, x_4	x_3	x_2
	x_1, x_3	x_4	x_2
	x_2, x_4	x_1	x_3
	x_1, x_4	x_2	x_3
	x_1, x_2	x_4	x_3
7	x_2, x_3	x_1	x_4
	x_1, x_3	x_2	x_4
8	x_1, x_2	x_3	x_4
	x_4	x_3	x_1, x_2
	x_3	x_4	x_1, x_2
	x_4	x_2	x_1, x_3
	x_2	x_4	x_1, x_3
	x_3	x_2	x_1, x_4
9	x_2	x_3	x_1, x_4

	x_1	x_4	x_2,x_3
	x_4	x_1	x_2,x_3
	x_3	x_1	x_2,x_4
	x_1	x_3	x_2,x_4
10	x_2	x_1	x_3,x_4
11	x_1	x_2	x_3,x_4
		x_2,x_3,x_4	x_1
12		x_1,x_3,x_4	x_2
		x_1,x_2,x_4	x_3
13		x_1,x_2,x_3	x_4
		x_3,x_4	x_1,x_2
		x_2,x_4	x_1,x_3
14		x_2,x_3	x_1,x_4
		x_1,x_4	x_2,x_3
		x_1,x_3	x_2,x_4
15		x_1,x_2	x_3,x_4
		x_4	x_1,x_2,x_3
16		x_2	x_1,x_3,x_4
17		x_1	x_2,x_3,x_4
		x_3	x_1,x_2,x_4
18			x_1,x_2,x_3,x_4

Operation 2

Here we calculate the maximal flow values of the networks corresponding to all possible combinations of states leaving the boundary. This is carried out by using the Maximal Flow computer program provided in Appendix B.

In the case where all states leave the boundary, there is no need to apply the computer program since, as stated in Part B of Section 2, the solution is trivial (see Figure 4.7).
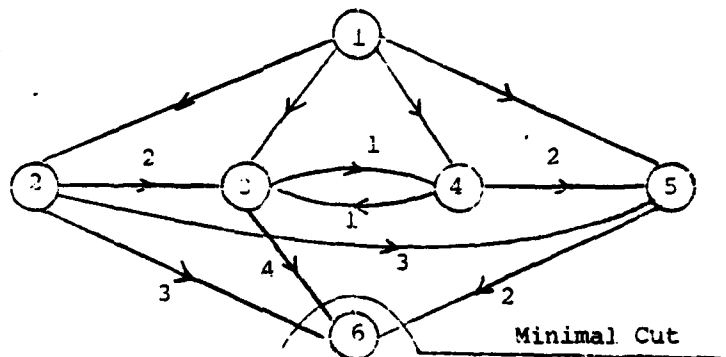
Figure 4.7 - Network representing all states leaving the boundary.

The solution is:

$$u_{1i} = u_{i6} = c_{i6} \ , \qquad i = 2,3,4,5$$

$$u_{ik} = 0 \qquad \qquad , \qquad k = 2,3,4,5$$

The value of the maximal flow is 9.

ARC	FLOW		CAPACITY
(1,3)	4		99
(1,4)	0		99
(1,5)	2		99
(2,3)	0		2
(2,5)	0		2
(2,6)	0		3
(3,4)	0		1
(3,6)	4	*	4
(4,3)	0		1
(4,5)	0		2
(5,6)	2	*	2
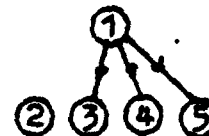
THE VALUE OF THE MAXIMAL FLOW IS   5

ARC	FLOW		CAPACITY
(1,2)	5		99
(1,4)	1		99
(1,5)	2		99
(2,3)	2	*	2
(2,5)	0		3
(2,6)	3	*	3
(3,4)	0		1
(3,6)	3		4
(4,3)	1	*	1
(4,5)	0		2
(5,6)	2	*	2
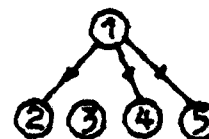
THE VALUE OF THE MAXIMAL FLOW IS   9

ARC	FLOW		CAPACITY
(1.2)	3		99
(1.3)	4		99
(1.5)	2		99
(2.3)	0		2
(2.5)	0		3
(2.6)	3	*	3
(3.4)	0		1
(3.6)	4	*	4
(4.3)	0		1
(4.5)	0		2
(5.6)	2	*	2
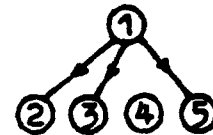
THE VALUE OF THE MAXIMAL FLOW IS   9

ARC	FLOW		CAPACITY
(1.2)	5		99
(1.3)	4		99
(1.4)	0		99
(2.3)	0		2
(2.5)	2		3
(2.6)	3	*	3
(3.4)	0		1
(3.6)	4	*	4
(4.3)	0		1
(4.5)	0		2
(5.6)	2	*	2

THE VALUE OF THE MAXIMAL FLOW IS   9

ARC	FLOW		CAPACITY
(1,4)	1		oo
(1,5)	2		99
(2,3)	0		2
(2,5)	0		3
(2,6)	0		3
(3,4)	0		1
(3,6)	1		4
(4,3)	1	*	1
(4,5)	0		2
(5,6)	2	*	2
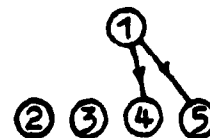
THE VALUE OF THE MAXIMAL FLOW IS   3

ARC	FLOW		CAPACITY
(1,3)	4		99
(1,5)	2		99
(2,3)	0		2
(2,5)	0		3
(2,6)	0		3
(3,4)	0		1
(3,6)	4	*	4
(4,3)	0		1
(4,5)	0		2
(5,6)	2	*	2

THE VALUE OF THE MAXIMAL FLOW IS   6

ARC	FLOW		CAPACITY
(1,3)	4		99
(1,4)	2		99
(2,3)	0		2
(2,5)	0		3
(2,6)	0		3
(3,4)	0		1
(3,6)	4	*	4
(4,3)	0		1
(4,5)	2	*	2
(5,6)	2		2
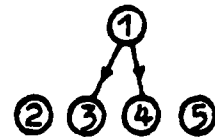
THE VALUE OF THE MAXIMAL FLOW IS 6

ARC	FLOW		CAPACITY
(1,2)	5		99
(1,5)	2		99
(2,3)	2	*	2
(2,5)	0		3
(2,6)	3	*	3
(3,4)	0		1
(3,6)	2		4
(4,3)	0		1
(4,5)	0		2
(5,6)	2	*	2
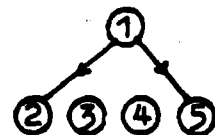
THE VALUE OF THE MAXIMAL FLOW IS 7

ARC	FLOW		CAPACITY
(1,2)	7		99
(1,4)	1		99
(2,3)	2	*	2
(2,5)	2		3
(2,6)	3	*	3
(3,4)	0		1
(3,6)	3		4
(4,3)	1	*	1
(4,5)	0		2
(5,6)	2	*	2
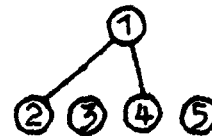
THE VALUE OF THE MAXIMAL FLOW IS   8

ARC	FLOW		CAPACITY
(1,2)	5		99
(1,3)	4		99
(2,3)	0		2
(2,5)	2		3
(2,6)	3	*	3
(3,4)	0		1
(3,6)	4	*	4
(4,3)	0		1
(4,5)	0		2
(5,6)	2	*	2

THE VALUE OF THE MAXIMAL FLOW IS   9

ARC	FLOW	CAPACITY
(1,5)	2	99
(2,3)	0	2
(2,5)	0	3
(2,6)	0	3
(3,4)	0	1
(3,6)	0	4
(4,3)	0	1
(4,5)	0	2
(5,6)	2	*	2
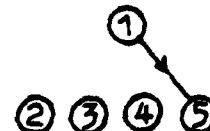
THE VALUE OF THE MAXIMAL FLOW IS   2

ARC	FLOW	CAPACITY
(1,4)	3	99
(2,3)	0	2
(2,5)	0	3
(2,6)	0	3
(3,4)	0	1
(3,6)	1	4
(4,3)	1	*	1
(4,5)	2	*	2
(5,6)	2	2
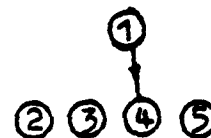
THE VALUE OF THE MAXIMAL FLOW IS   3

ARC	FLOW		CAPACITY
(1,3)	5		99
(2,3)	0		2
(2,5)	0		3
(2,6)	0		3
(3,4)	1	*	1
(3,6)	4	*	4
(4,3)	0		1
(4,5)	1		2
(5,6)	1		2

THE VALUE OF THE MAXIMAL FLOW IS   5

ARC	FLOW		CAPACITY
(1,2)	7		99
(2,3)	2	*	2
(2,5)	2		3
(2,6)	3	*	3
(3,4)	0		1
(3,6)	2		4
(4,3)	0		1
(4,5)	0		2
(5,6)	2	*	2
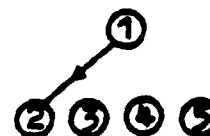
THE VALUE OF THE MAXIMAL FLOW IS   7

The links belonging to a minimal cut are marked by asterisks (*) in each of the above programs.

Now we summarize the results we obtained:

D	k(D)
x_1, x_2, x_3, x_4	9
x_1, x_2, x_3	9
x_1, x_2, x_4	9
x_1, x_2	9
$x_1, \bar{x_3}, x_4$	8
x_1, x_3	8
x_1, x_4	7
x_1	7
x_2, x_3, x_4	6
x_2, x_3	6
x_2, x_4	6
x_2	5
x_3, x_4	3
x_3	3
x_4	2

This table will assist us in the execution of Operations 3, 4 and 5.

Operation 3

In this operation we erase redundant sequences from the list of Operation 1. The Operation is made by searching all the sequences of the list and by checking if the conditions of redundancy are satisfied considering the results obtained in Operation 2.

In our example we obtain that only 18 sequences from the 75 listed in Operation 1 are nonredundant. These sequences are marked and numbered in the list of Operation 1.

Operation 4

Here we find all basic optimal solutions for all combinations of states leaving the boundary. By the summarizing table of the maximal flow values we realize that it is enough to find all the solutions for only five combinations of states. Notice that in the cases where only one state leaves the boundary the solution is unique, and it is given by the result obtained in Operation 2. The five networks for which we have to find all optimal solutions correspond to the largest combinations of states (containing at least two states) having different maximal flow values.

Using the results of Operation 2 (where we localize minimal cuts) and by the considerations of Part C of Section 2, we obtain first basic optimal solutions to every one of the networks, as shown in Figure 4.8.

(a)

(b)

(c)

(d)

(e)
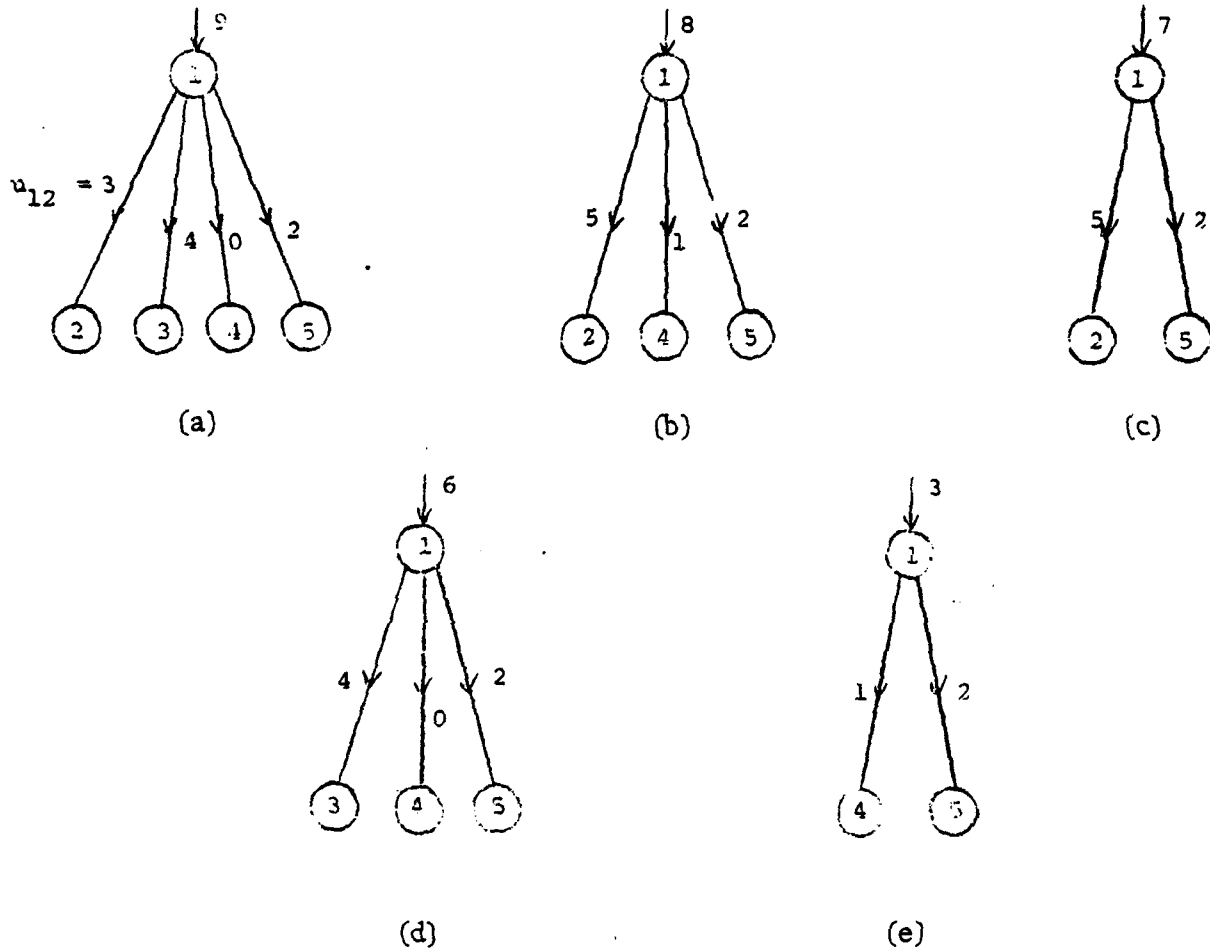
Figure 4.8 - The first optimal solutions

Applying the method for finding all the solutions described in Part B
of this Section (the computer program implementing it is provided in Appendix B),
on the above networks, we obtain the following results:

(1,2)	(1,3)	(1,4)	(1,5)	(2,3)	(3,4)	(4,3)	(4,5)	(2,5)
3	4	0	2	0	0	0	0	0
5	2	0	2	2	0	0	0	0
3	3	1	2	0	0	1	0	0
3	4	2	0	0	0	0	2	0
5	4	0	0	0	0	0	0	2
5	1	1	2	2	0	1	0	0
5	2	2	0	2	0	0	2	0
7	2	0	0	2	0	0	0	2
3	4	0	2	0	1	1	0	0
3	5	0	1	0	1	0	1	0
3	3	3	0	0	0	1	2	0
5	3	1	0	0	0	1	0	2
3	5	1	0	0	1	0	2	0
5	2	0	2	2	1	1	0	0
5	3	0	1	2	1	0	1	0
5	1	3	0	2	0	1	2	0
7	1	1	0	2	0	1	0	2
5	3	1	0	2	1	0	2	0
5	4	0	0	0	1	1	0	2
4	5	0	0	0	1	0	1	1
3	4	2	0	0	1	1	2	0
7	2	0	0	2	1	1	0	2
6	3	0	0	2	1	0	1	1
5	2	2	0	2	1	1	2	0

NUMBER OF DIFFERENT SOLUTIONS :   24

NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS   :   45

(a)

1.0

1.1

1.25

1.4

1.6

1.8

2.0

2.2

2.5

2.8

3.2

3.6

4.0

4.5

5.0

5.6

6.3

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU
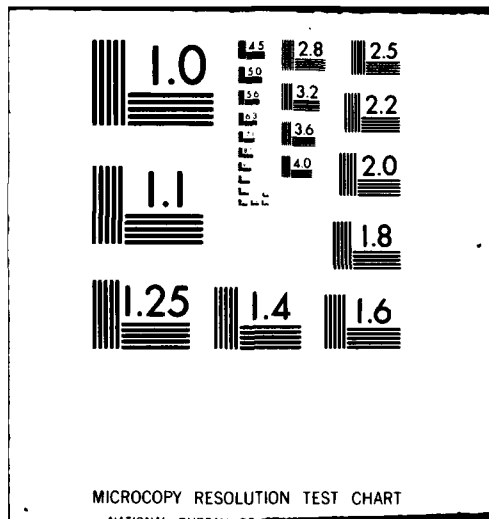
(1,2)	(1,4)	(1,5)	(4,5)	(2,5)
5	1	2	0	0
5	2	0	2	0
7	1	0	0	2

NUMBER OF DIFFERENT SOLUTIONS :   3

NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS :   3

(b)

(1,2)	(1,5)	(2,5)
5	2	0
7	0	2

NUMBER OF DIFFERENT SOLUTIONS :   2

NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS :   2

(c)

(1,3)	(1,4)	(1,5)	(3,4)	(4,3)	(4,5)
4	0	2	0	0	0
3	1	2	0	1	0
4	2	0	0	0	2
4	0	2	1	1	0
5	0	1	1	0	1
3	3	0	0	1	2
5	1	0	1	0	2
4	2	0	1	1	2

NUMBER OF DIFFERENT SOLUTIONS :   4

NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS :   14

(d)

(1,4)	(1,5)	(4,5)
1	2	0
3	0	2

NUMBER OF DIFFERENT SOLUTIONS :   2

NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS :   2

(e)

Operation 5

In this Operation, the feedback control regions are constructed with the aid of the results of Operation 4, and by calculating the costate values in every interval of every (non-redundant) sequence. In this example we restrict ourselves to finding the costate values since the remaining steps do not represent interesting features of the algorithm.

As we said in Part A of Section 4, when developing the algorithm it turns out that the calculation of the costate values is almost trivial for the non-redundant sequences. In the following table we rewrite these sequences where under every interval we write the corresponding costate vector $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. We have taken arbitrarily $t_{p+1} - t_p = 1$, and the states on every interval correspond to I_p, that is states travelling on interior arcs.

	t_{f-4}	t_{f-3}	t_{f-2}	t_{f-1}
1	x_1, x_2, x_3, x_4	x_1, x_3, x_4	x_1, x_4	x_4
	$(3,1,2,4)$	$(2,0,1,3)$	$(1,0,0,2)$	$(0,0,0,1)$
2	x_1, x_2, x_3, x_4	x_1, x_3, x_4	x_3, x_4	x_4
	$(2,1,3,4)$	$(1,0,2,3)$	$(0,0,1,2)$	$(0,0,0,1)$
3	x_1, x_2, x_3, x_4	x_2, x_3, x_4	x_3, x_4	x_4
	$(1,2,3,4)$	$(0,1,2,3)$	$(0,0,1,2)$	$(0,0,0,1)$
4		x_1, x_2, x_3, x_4	x_2, x_3, x_4	x_2
		$(1,3,2,2)$	$(0,2,1,1)$	$(0,1,0,0)$
5		x_1, x_2, x_3, x_4	x_2, x_3, x_4	x_4
		$(1,2,2,3)$	$(0,1,1,2)$	$(0,0,0,1)$
6		x_1, x_2, x_3, x_4	x_1, x_3, x_4	x_4
		$(2,1,2,3)$	$(1,0,1,2)$	$(0,0,0,1)$

7	x_1,x_2,x_3,x_4 $(2,1,1,3)$	x_1,x_4 $(1,0,0,2)$	x_4 $(0,0,0,1)$	
8	x_1,x_2,x_3,x_4 $(1,1,2,3)$	x_3,x_4 $(0,0,1,2)$	x_4 $(0,0,0,1)$	
9	x_1,x_2,x_3,x_4 $(3,1,2,3)$	x_1,x_3,x_4 $(2,0,1,2)$	x_1,x_4 $(1,0,0,1)$	
10	x_1,x_2,x_3,x_4 $(2,1,3,3)$	x_1,x_3,x_4 $(1,0,2,2)$	x_3,x_4 $(0,0,1,1)$	
11	x_1,x_2,x_3,x_4 $(1,2,3,3)$	x_2,x_3,x_4 $(0,1,2,2)$	x_3,x_4 $(0,0,1,1)$	
12		x_1,x_2,x_3,x_4 $(1,2,1,1)$	x_2 $(0,1,0,0)$	
13		x_1,x_2,x_3,x_4 $(1,1,1,2)$	x_4 $(0,0,0,1)$	
14		x_1,x_2,x_3,x_4 $(2,1,1,2)$	x_1,x_4 $(1,0,0,1)$	
15		x_1,x_2,x_3,x_4 $(1,1,2,2)$	x_3,x_4 $(0,0,1,1)$	
16		x_1,x_2,x_3,x_4 $(2,1,2,2)$	x_1,x_3,x_4 $(1,0,1,1)$	
17		x_1,x_2,x_3,x_4 $(1,2,2,2)$	x_2,x_3,x_4 $(0,1,1,1)$	
18			x_1,x_2,x_3,x_4 $(1,1,1,1)$	

Section 5

CONCLUSIONS

We presented a new approach for the construction of a feedback solu-
tion to the minimal delay dynamic message routing problem for single destina-
tion networks with all unity weightings in the cost functional. The approach
seems to be the most appropriate one for tackling the problem, since it fully
exploits the special structure of the constraints matrix and also provides a
physical meaning to the problem by working in a framework of networks rather
than in an abstract mathematical one.

Several improvements have been achieved compared to previous results.
The first is the great reduction in the number of linear programs to be solved
by taking advantage of the high degree of degeneracy that in general character-
izes this kind of problems. The second is the derivation of a suitable method
for finding all solutions to the linear programs, that does not require the
application of simplex techniques to achieve a first optimal solution but only
the application of a simple algorithm of maximal flow. Moreover, the method
reaches all the remaining optimal solutions by pivoting operations on a net-
work, saving in this way a great amount of computer memory. The third improve-
ment achieved is that the new approach provides the tools for analyzing the
complexity of the problem, giving us an idea of the number of computational
steps required for obtaining the feedback solution to a given network.

Further research is required for Operation 1 of the algorithm. Since
not all sequences listed in the above Operation are used for constructing
feedback control regions, the question is whether a method can be provided to
avoid the listing of those sequences not contributing to the construction of

the feedback solution. It also seems that the algorithm may be applicable in the case when constant inputs are present. However, further research is needed to investigate the influence of these inputs on the different Operations of the algorithm.

Finally, it also remains to extend the application of the approach to the more general case of multidestination networks.

Appendix A

Basic Concepts of

Graph Theory, Maximal Flow and Linear Programming in Networks

1. Basic Definitions

Directed Graph: A set of nodes $N = \{1, 2, \ldots, n\}$ and a set of directed links $L = \{(i,j), (k,\ell), \ldots, (v,w)\}$ connecting pairs of nodes of N.

Path: A series of different nodes in the graph where between two successive nodes of the series there is a link connecting them.

Cycle: A path to which a link is added connecting the first and the last node on it.

Connected Graph: A graph in which there is at least one path between any pair of nodes.

Tree: A connected graph not containing cycles.

Spanning Tree: A tree containing all the nodes of the graph.

Network: A connected graph in which to every link (i,j) corresponds a positive integer c_{ik} called the capacity of the link.

2. Maximal Flow on Networks (see [H1], [B3]).

We define two special nodes of the network; the first is the source (s) and the second the destination (d).

The concept flow on the network is defined as follows: A set of non-negative integers u_{ik} is called flow in the network if the following constraints are satisfied:

$$\sum_i u_{ij} - \sum_k u_{jk} = \begin{cases} -f & \text{if } j = s \\ 0 & \text{if } j \neq s,d \\ f & \text{if } j = d \end{cases} \qquad \text{(A.1)}$$

$$0 \leq u_{ij} \leq c_{ij} , \quad \forall (i,j) \, \epsilon \, L . \qquad \text{(A.2)}$$

The <u>Maximal Flow Problem</u> is defined as:

$$\text{Maximize } f \qquad \text{(A.3)}$$

such that (A.1) and (A.2) are satisfied.

A <u>Cut</u> separating the nodes s and d is denoted by $(X,\bar{X})$ and defined as follows:

$$(X,\bar{X}) = \{(i,j) \, \epsilon \, L \, / \, i \, \epsilon \, X, j \, \epsilon \, \bar{X}\} ,$$

where X is a set of nodes in the network containing the node s but not the node d, and $\bar{X} = N/X$.

The <u>Cut Value</u> is

$$\sum_{\substack{j \epsilon X \\ j \epsilon \bar{X}}} c_{ij} .$$

A <u>Minimal Cut</u> of the network is defined as a cut having minimal value.

<u>Theorem</u> (The Maximal Flow-Minimal Cut, (see [F1]))

The maximal flow in a network, between the source and the destination equals the value of a minimal cut separating the source and the destination.

From the above Theorem follows that if $(X_M, \bar{X}_M)$ is a minimal cut of the network, then at maximal flow we have

$$u_{ik} = c_{ik} \quad \forall \{(i,j)/i \, \epsilon \, X_M, j \, \epsilon \, \bar{X}_M\}$$

$$u_{k\ell} = 0 \quad \forall \{(k,\ell)/k \, \epsilon \, \bar{X}_M, \ell \, \epsilon \, X_M\} .$$

3. Minimal Cost Network Flows (see [B3],[H1]). ·

Consider a network having n nodes and m links. To every node i in the network corresponds an integer b_i that accounts for either the "supply" of flow to the node (if $b_i > 0$), or the "demand" of flow from the node (if $b_i < 0$). Assume also that the total supply of flow to the network equals the total demand from it, that is $\sum_{i=1}^{n} b_i = 0$. To every link (i,k) in the network corresponds a number $\gamma_{ij} \geq 0$ that expresses the cost per unity of flow on the link.

The Minimal Cost Network Flow Problem is the determination of the flows on the links such that the supply of flow to the network satisfies the demand of flow from it at minimal cost. Mathematicaly the problem is stated as follows (the summations are taken for existing links):

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \gamma_{ij} u_{ij} \quad , \tag{A.4}$$

such that

$$\sum_{j=1}^{n} u_{ij} - \sum_{k=1}^{n} u_{ki} = b_i \quad , \quad i = 1,2,\ldots,n \quad , \tag{A.4a}$$

$$0 \leq u_{ij} \leq c_{ij}, \quad \forall (i,j) \in L \quad . \tag{A4.b}$$

The constraints (A.4a), written in vector form, are $A\underline{u} = \underline{b}$.

Now we investigate the structure of the constraint matrix A: Every row of A corresponds to a node and every column to a link, thus the dimensions of the matrix is $(n \times m)$. Every column of A has only two non-zero elements, +1 in the i-th row and -1 in the j-th row, where i and j are the exit and entrance nodes of the link, respectively. The rank of the A-matrix is $(n-1)$. To see this, note that every row in A accounts for the flow conservation on the corresponding node, so that the row corresponding to node r is given by the sum of all the remaining rows of A with minus sign.

These remaining rows are clearly independent since the deletion of the r-th row from A leaves at least one column having only one non-zero element.

Now we shall deal with the linear programming problem (A.4), when the links have no capacity constraints, that is we require only $u_{ij} \geq 0$. In a linear programming problem, the variables corresponding to some of the columns of the constraint matrix are called nonbasic variables, or independent variables, because their values are determined arbitrarily. The values of the remaining variables (the basic or dependent variables) are determined by means of the non-basic variables and the constraint matrix. The columns corresponding to the basic variables form a square matrix B, called the basic matrix, and if these columns are chosen such that they are linearly independent, and if the variables not associated with columns of B are set equal to zero, then the solution of the system $B\underline{u} = \underline{b}$ is unique (since B is non-singular), and the above solution is an extreme point of the constraint figure. In the case of problem (A.4), it is easy to see that from all links incident to a node, the flow on one of them cannot be determined arbitrarily but by the flow conservation equation of the node. If we consider all links on which the flows are determined by the remaining links of the network, these links form a spanning tree. The reason for that is that as we said before, the flow on one of the links incident to a node is determined uniquely by the flows on the remaining links incident to the node satisfying in this way the flow conservation equation of the node. Now the constraint matrix A has (n-1) linearly independent rows, that is, there are (n-1) independent flow conservation equations, so that there are (n-1) links in the network on which the flow is determined by the flows on the remaining links. These (n-1) links cannot form any cycles because one can add flows in the cycle without violating the flow conservation equations of the nodes in the cycle. This would contradict the fact that the values of these (n-1) links are uniquely

determined. Moreover, to every node corresponds at least one such link, there-
fore the links with flows representing basic variables form a spanning tree.

Next we describe how to represent a nonbasic vector (that is, a column
of A corresponding to a nonbasic variable) in terms of basic vectors. In order
to do this, note that every column of A (denote it by a_{ij}) is of the form

$$a_{ij} = e_i - e_j \quad ,$$

where e_i and e_j are unity vectors in E^n, the former having a +1 on the
i-th row and the latter having a +1 on the j-th row. Now, by choosing (n-1)
linearly independent columns of A we form matrix B. Matrix B represents a
spanning tree in the network. Now if we choose a nonbasic link, say the link
(v,w), clearly we will have a unique path consisting of basic links only,
connecting the nodes v and w, since the basic links of the network form
a spanning tree. The above path together with the nonbasic link (u,w) *form*
a cycle (see Figure A1). By determining the cycle orientation as the direc-
tion of the nonbasic link we will have

$$a_{vw} - a_{vj} + a_{kj} + \ldots + a_{wp} = (e_v - e_w) - (e_v - e_j) + (e_k - e_j) + \ldots + (e_w - e_p) = 0 \quad ,$$

or

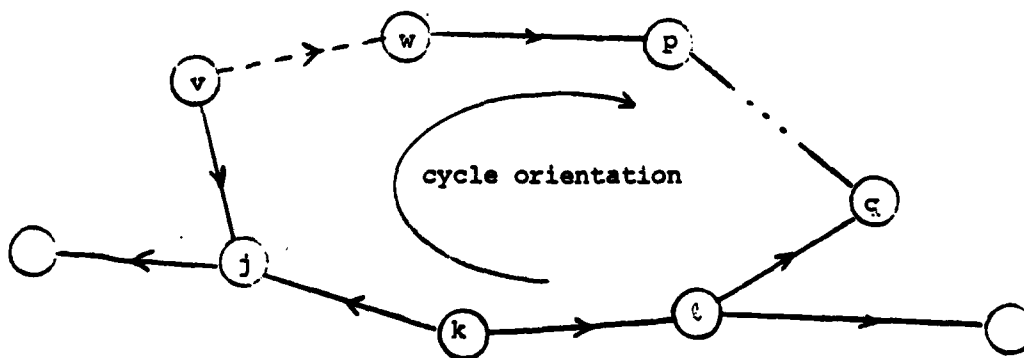$$a_{vw} = a_{vj} - a_{kj} + \ldots + a_{wp} \quad .$$

Figure A1 - Cycle formed by adding a nonbasic link to the spanning tree.

The method for representing a nonbasic vector in terms of basic vectors is then as follows: First, we determine the cycle formed by adding the nonbasic link to the spanning tree, Second, we assign to the cycle an orientation, according to the direction of the nonbasic link. Third, to all basic links in the cycle having the same direction of the cycle we assign a -1 coefficient in the representation and to all basic links in the cycle with direction opposite to that of the cycle we assign a +1 coefficient in the representation. To all remaining links in the spanning tree we assign a zero coefficient. The above coefficients correspond to those appearing in the columns of the Simplex tableau. As said before, these coefficients are +1, -1 or zero. This property of the A-matrix is called the unimodularity property and it insures that every basic solution is formed by integers only, provided that $\underline{b}$ is an all-integers vector.

From the above considerations it turns out that in order to perform a pivot in problem (A.4), it is enough to form a cycle and to increase the flow on every link of the cycle in the direction of the nonbasic link, until the flow on one of the basic links becomes zero. This link leaves the basis and the previous nonbasic link enters the basis. When there is a basic link in the cycle, with direction opposite to it and with zero flow, we say that the cycle is degenerate. In this case we cannot perform a pivot changing the flow on the cycle, but we can obtain another representation of the current basic solution by removing the link that does not permit flow change from the basis and putting the nonbasic link into the basis. If there are two or more links yielding degeneracy, we can reach all possible representations by applying the above method.

Next we return to problem (A.4), but now consider the case where the links do have capacity constraints. It turns out that the problem can be solved in a manner very similar to the one used in the case without capacity

constraints by making use of the concept "extended basic solution" (see [L1],
p. 48). The idea is to treat the capacity constraints (or upper bound con-
straints) in an implicit way (similar to non-negative constraints on variables).
This method avoids the great increase of dimensionality in the problem, result-
ing from the addition of slack variables to the upper bound constraints. For
example, if in problem (A.4) the A-matrix has dimension $(n \times m)$, then adding
a slack variable to each of the constraints (A.4b) yields to a constraint
matrix of dimension $(n + m) \times 2m$. The concept "extended basic solution" cor-
responding to problem (A.4) is defined as a feasible solution in which n
variables corresponding to linearly independent columns of A are basic, and
the remaining $(m - n)$ variables are nonbasic, each having either value zero
or being equal to its upper bound (i.e. its capacity). In the network, an
"extended basic solution" is characterized by a spanning tree, that is, the
graph formed by the links corresponding to basic variables is a spanning tree.
Each link corresponding to a nonbasic variable carries either no flow, or flow
equal to the capacity of the link.

Starting from an initial "extended basic solution", in order to make
a pivot in the network, we choose one of the links that is nonbasic and look
for a cycle formed by it with basic type links. If the flow on the nonbasic
link is zero, then we check if we can increase the flow on every link of the
cycle in the direction of the nonbasic link. If the flow on the nonbasic
link is equal to its capacity, then we try to increase the flow on every link
of the cycle in the direction opposite to that of the nonbasic link. In both
cases we increase the flow until either:

(i) The flow on a basic link of the cycle reaches the value of zero or its
     capacity.

(ii) The flow on the nonbasic link reaches the value of its capacity
     (or zero).

If (i) occurs first, then the basic link is declared nonbasic, and the previous nonbasic link is declared basic. If (ii) occurs first, then the spanning tree does not change and the nonbasic link remains nonbasic with another flow.

A degenerate situation corresponds to the case where there is at least one basic link in the cycle with direction opposite to that of the cycle and with zero flow, or a basic link in the direction of the cycle, with flow equal to its capacity.

Appendix B

Computer Programs

1. Maximal Flow

The Maximal Flow Algorithm of Edmons and Karp is implemented by a Fortran Subroutine called MAXFL. The algorithm finds the shortest path between source and destination on which an increase of flow is feasible, determines the maximal amount of flow that can be pushed on the path and then performs the increase of flow. The algorithm stops when no such path can be found, and localizes the minimal cut that is nearest to the source.

Input Data

N - Number of nodes in the network.

C - Two-dimensional array for the link capacities. Capacity zero corresponds to non-existing links. To links not having capacity constraints we assign a very large capacity (at least as the sum of the capacities of the exit links from the corresponding node). The dimension of C is (N,N).

Program Output

The program writes the flow on every link corresponding to maximal flow and the link capacity. In addition it marks with an asterisk the links directed to the destination that belongs to the minimal cut, and writes the maximal flow value.

Dimensioned Local Variables

F - Two-dimensional array for the link flows. The dimension is (N,N). XT, LAB, NODE and BNODE are one-dimensional arrays of dimension N.

```fortran
C
C     MAIN PROGRAM
      INTEGER C(6,6),F(6,6),XT(6)
      COMMON C,F,XT,LAB(6),L,K,N,MFL
      DO 1000 LL=1,14
      READ(5,10) N,((C(I,J),J=1,N),I=1,N)
      DO 1 I=1,N
      DO 1 J=1,N
1     F(I,J)=0
      DO 2 J=1,N
      IF(C(1,J))101,2,101
101   IF(C(J,N))102,2,102
102   F(1,J)=C(J,N)
      F(J,N)=C(J,N)
2     CONTINUE
      CALL MAXFL
      WRITE(6,14)
      DO 3 I=1,N
      DO 3 J=1,N
      IF(C(I,J))103,3,103
103   DO 4 II=1,K
      IF(I-LAB(II))4,105,4
4     CONTINUE
      GO TO 106
105   DO 5 J1=1,L
      IF(J-XT(J1))5,107,5
5     CONTINUE
106   WRITE(6,11) I,J,F(I,J),C(I,J)
      GO TO 3
107   WRITE(6,12) I,J,F(I,J),C(I,J)
3     CONTINUE
      WRITE(6,13) MFL
11    FORMAT(/,9X,'(',I1,',',I1,')',7X,I2,7X,I2)
12    FORMAT(/,9X,'(',I1,',',I1,')',7X,I2,3X,'*',3X,I2)
13    FORMAT(///,2X,'THE VALUE OF THE MAXIMAL FLOW IS',I4)
14    FORMAT(1H1,9X,'ARC',8X,'FLOW',3X,'CAPACITY')
10    FORMAT(36I2/36I2)
1000  CONTINUE
      STOP
      END
```

```
          SUBROUTINE MAXFL
          INTEGER C(6,6),F(6,6),XT(6),NODE(6),BNODE(6),DELTA
          COMMON C,F,XT,LAB(6),L,K,N,MFL
107       I=1
          K=0
          L=0
          DO 7 J=1,N
          NODE(J)=0
7         BNODE(J)=0
          NODE(1)=90
118       DO 1 J=1,N
          K1=NODE(I)
          DELTA=IFIX(ABS(FLOAT(K1)))
          IF(C(I,J))100,100,101
101       IF(C(I,J)-F(I,J))100,100,102
102       IF(NODE(J))1,103,1
103       K=K+1
          LAB(K)=J
          IF(C(I,J)-F(I,J)-DELTA)105,104,104
105       DELTA=C(I,J)-F(I,J)
104       NODE(J)=DELTA
          BNODE(J)=I
          IF(J-N)1,121,1
121       J1=J
106       IF(J1-1)108,107,108
108       I=BNODE(J1)
          IF(NODE(J1))110,110,109
109       F(I,J1)=F(I,J1)+DELTA
120       J1=I
          GO TO 106
110       F(J1,I)=F(J1,I)-DELTA
          GO TO 120
100       IF(C(J,I))1,1,111
111       IF(F(J,I))1,1,112
112       IF(NODE(J))1,113,1
113       K=K+1
          LAB(K)=J
          IF(F(J,I)-DELTA)114,114,115
114       DELTA=F(J,I)
115       NODE(J)=-DELTA
          BNODE(J)=I
1         CONTINUE
          L=L+1
          IF(K-L)116,117,117
117       I=LAB(L)
          GO TO 118
116       L=0
          MFL=0
          DO 3 I=2,N
          DO 4 J=1,K
          IF(LAB(J)-I)4,3,4
4         CONTINUE
          L=L+1
          XT(L)=I
3         CONTINUE
          DO 5 I=1,K
          DO 5 J=1,L
          M=LAB(I)
          N1=XT(J)
          IF(C(M,N1))119,5,119
119       MFL=MFL+F(M,N1)
5         CONTINUE
          RETURN
          END
```

2. Determination of all Optimal Solutions

The algorithm described in Part B of Section 4 is implemented by a
Fortran Computer Program composed of two subroutines and a main program.

General Flow Diagram

```
┌─────────────────────────────┐
│        MAIN PROGRAM         │
│         Reads data          │──┐
│   Chooses a nonbasic link   │  │
│   Filters representations   │  │
│        Writes results       │  │
└──────────────┬──────────────┘  │
               │                 │
┌──────────────┴──────────────┐  │
│      SUBROUTINE CYCLE        │  │
│       Finds the cycle        │  │
│    corresponding to the      │  │
│     nonbasic link chosen     │  │
│     in the main program.     │  │
└──────────────┬──────────────┘  │
               │                 │
┌──────────────┴──────────────┐  │
│      SUBROUTINE PIVOT        │  │
│     Finds the permissible    │  │
│   change of flow on the cycle.│ │
│   Performs pivot to obtain a │  │
│    new optimal solution or   │──┘
│        representation.       │
└─────────────────────────────┘
```
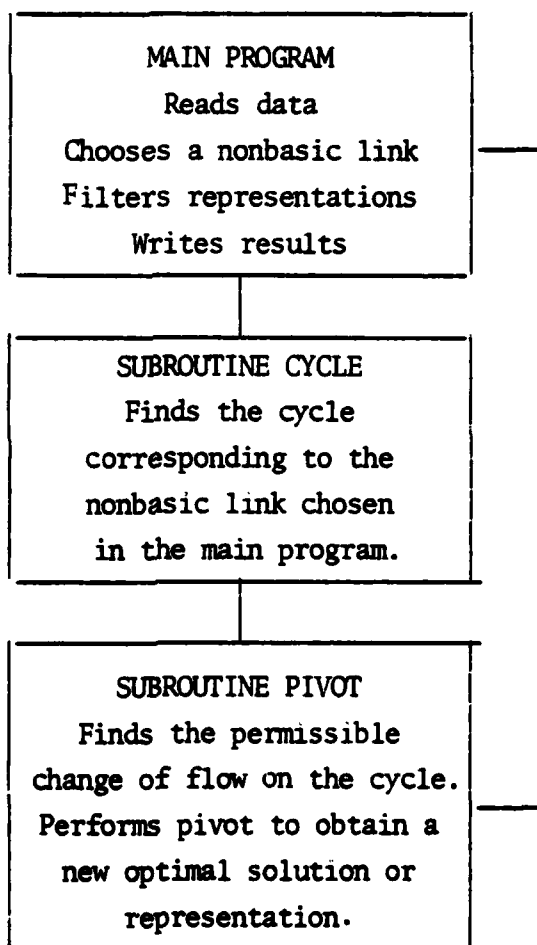
Input Data

    M - Number of network links.

    C - One-dimensional array for the link capacities. The dimension
is M.

    F - Two dimensional array for the flows on the links at the current
optimal solution. Its dimension is (M,N), where N is the

estimated number of different solutions and representations. A
first optimal solution must be supplied to the program (that
obtained from the maximal flow algorithm) that is $F(I,1)$.

BASE - Two-dimensional array that determines which links are basic and
which nonbasic at the current solution. Its dimension is (M,N).
If link I is basic then BASE $(I,\cdot) = 1$, if it is nonbasic
then BASE $(I,\cdot) = 0$. The basic configuration of the first optimal
solution must be provided to the program, that is BASE$(I,1)$.

LOUT - One-dimensional array for the exit nodes of the links.

LIN - One-dimensional array for the entrance nodes of the links.

Program Output

The program writes all the extremal optimal solutions and their number.
In addition it writes the number of different solutions and representations
searched during the execution.

Dimensioned Local Variables

DIRF, BFR and LAB are one-dimensional arrays of dimension M.

```
C          DETERMINATION OF ALL OPTIMAL SOLUTIONS - MAIN PROGRAM
           INTEGER C(9),F(9,50),BASE(9,50),DIRF(9),BFR(9),DELT
           COMMON K,K1,K2,DELT,J1,I1,C,LOUT(9),LIN(9),F,BASE,DIRF,BFR,M,NR
           READ(5,10) M
           READ(5,11)  (C(I),I=1,M)
           READ(5,11)  (F(I,1),I=1,M)
           READ(5,11)  (BASE(I,1),I=1,M)
           READ(5,11)  (LOUT(I),I=1,M)
           READ(5,11)  (LIN(I),I=1,M)
           WRITE(6,13)  (LOUT(I),LIN(I),I=1,M)
           WRITE(6,14)  (F(I,1),I=1,M)
           K=1
           K1=K
           NP=1
           I2=1
1          J1=0
2          J1=J1+1
           IF(J1.GT.M)GO TO 3
           IF (BASE(J1,K).GT.0)GO TO 2
           CALL CYCLE
           DO 100 I=1,K2
           DO 200 J=1,M
           IF(F(J,K1).NE.F(J,I))GO TO 100
200        CONTINUE
           GO TO 2
100        CONTINUE
           WRITE(6,14)  (F(I,K1),I=1,M)
           I2=I2+1
           GO TO 2
3          K=K+1
           IF(K1.GE.K)GO TO 1
           WRITE(6,15) I2
           WRITE(6,16) NR
10         FORMAT(I2)
11         FORMAT(36I2)
13         FORMAT(1H1,3X,9('(',I1,',',I1,')',3X))
14         FORMAT(/,5X,9(I2,6X))
15         FORMAT(///,2X,'NUMBER OF DIFFERENT SOLUTIONS :',I5)
16         FORMAT(///,2X,'NUMBER OF DIFFERENT SOLUTIONS AND REPRESENTATIONS
          *:',I5)
           DEBUG SUBCHK
           STOP
           END

C************************************************************************
           SUBROUTINE PIVOT
           INTEGER C(9),F(9,50),BASE(9,50),DIRF(9),BFR(9),DELT
           COMMON K,K1,K2,DELT,J1,I1,C,LOUT(9),LIN(9),F,BASE,DIRF,BFR,M,NR
           K1=K1+1
           DO 100 I=1,M
100        F(I,K1)=F(I,K)
           I2=I1
1          IF(DIRF(I1).EQ.1)GO TO 2
           F(I1,K1)=F(I1,K1)-DELT
           GO TO 3
2          F(I1,K1)=F(I1,K1)+DELT
3          IF(I1.EQ.J1)GO TO 4
           I1=BFR(I1)
           GO TO 1
4          IF(F(I2,K1).EQ.C(I2))GO TO 5
           IF(F(I2,K1).EQ.0)GO TO 5
           I2=BFR(I2)
           GO TO 4
5          DO 200 I=1,M
200        BASE(I,K1)=BASE(I,K)
           BASE(J1,K1)=1
           BASE(I2,K1)=0
           K2=K1-1
           DO 400 I=1,K2
           DO 300 J=1,M
           IF(F(J,K1).NE.F(J,I))GO TO 400
           IF(BASE(J,K1).NE.BASE(J,I))GO TO 400
300        CONTINUE
           K1=K1-1
           RETURN
400        CONTINUE
           NR=NR+1
           RETURN
           END
```

```
C***********************************************************************
      SUBROUTINE CYCLE
      INTEGER C(9),F(9,50),BASE(9,50),DIRF(9),BFR(9),DELT,LAB(9)
      COMMON K,K1,K2,DELT,J1,I1,C,LOUT(9),LIN(9),F,BASE,DIRF,BFR,M,NP
      L=0
      M1=1
      M2=1
      J2=J1
      DO 100 I=1,M
100   LAB(I)=0
      DIRF(J1)=1
      IF(F(J1,K).NE.C(J1))GO TO 25
      M2=2
      DIRF(J1)=-1
      M1=2
25    J=0
1     I=1
2     IF(I.GT.M)GO TO 4
      GO TO (10,11),M1
11    IF(LOUT(I).NE.LOUT(J2))GO TO 3
      GO TO 12
10    IF(LOUT(I).NE.LIN(J2))GO TO 3
12    IF(BASE(I,K).NE.1)GO TO 3
      DO 200 N=1,M
      IF(LAB(N).EQ.I)GO TO 3
200   CONTINUE
      DIRF(I)=1
      J=J+1
      LAB(J)=I
      BFR(I)=J2
      GO TO (21,22),M2
21    IF(LIN(I).EQ.LOUT(J1))GO TO 8
      GO TO 3
22    IF(LIN(I).EQ.LIN(J1))GO TO 8
3     I=I+1
      GO TO 2
4     I=1
5     IF(I.GT.M)GO TO 7
      GO TO (13,14),M1
14    IF(LIN(I).NE.LOUT(J2))GO TO 6
      GO TO 15
13    IF(LIN(I).NE.LIN(J2))GO TO 6
15    IF(BASE(I,K).NE.1)GO TO 6
      DO 300 N=1,M
      IF(LAB(N).EQ.I)GO TO 6
300   CONTINUE
      DIRF(I)=-1
      J=J+1
      LAB(J)=I
      BFR(I)=J2
      GO TO (23,24),M2
23    IF(LOUT(I).EQ.LOUT(J1))GO TO 9
      GO TO 6
24    IF(LOUT(I).EQ.LIN(J1))GO TO 9
6     I=I+1
      GO TO 5
7     L=L+1
      J2=LAB(L)
      M1=1
      IF(DIRF(J2).EQ.-1)M1=2
      GO TO 1
8     I1=I
      DELT=C(I)-F(I,K)
      GO TO 16
9     I1=I
      DELT=F(I,K)
16    J3=BFR(I)
      IF(DIRF(J3).EQ.1)GO TO 17
      IF(DELT.LT.F(J3,K))GO TO 18
      DELT=F(J3,K)
      GO TO 18
17    IF(DELT.LT.(C(J3)-F(J3,K)))GO TO 18
      DELT=C(J3)-F(J3,K)
18    IF(J3.EQ.J1)GO TO 20
      I=J3
      GO TO 16
20    CALL PIVOT
      RETURN
```

References

[B1] Balinski, M.L., "An Algorithm for Finding All the Vertices of Convex Polyhedral Sets", _J. Soc. Indust. Appl. Math._, Vol. 9, No. 1, March 1961.

[B2] Bloom, J., "A Program for Chernikova's Algorithm", Electronic Systems Laboratory, M.I.T., March 1976.

[B3] Bazaraa, M.S. and Jarvis, J.J., _Linear Programming and Network Flows_, John Wiley & Sons, 1977.

[Ch1] Chernikova, N.V., "Algorithm for Finding a General Formula for the Non-negative Solutions of a System of Linear Equations", _U.S.S.R. Computational Mathematics and Mathematical Physics_, 4, pp. 151-158, 1964.

[Ch2] Chernikova, N.V., "Algorithm for Finding a General Formula for the Non-negative Solutions of a System of Linear Inequalities", _U.S.S.R. Computational Mathematics and Mathematical Physics_, 5, pp pp. 228-233, 1965.

[E1] Even, S., _Algorithmic Combinatorics_, The MacMillan Company, N.Y., 1973.

[F1] Ford, L.R., Jr., and Fulkerson, D.R., "Maximal Flow Through a Network", _Canadian J. Math._, 8 (3), pp. 339-404, 1950.

[H1] Hu, T.C., _Integer Programming and Network Flows_, Addison-Wesley, 1970.

[L1] Luenberger, D.G., _Introduction to Linear and Nonlinear Programming_, Addison-Wesley, 1973.

[M1] Moss, F.H., "The Application of Optimal Control Theory to Dynamic Routing in Data Communication Networks", _Ph.D. Dissertation_, Massachusetts Inst. Technol., Cambridge, 1976.

[M2] Moss, F.H. and Segall, A., "An Optimal Control Approach to Dynamic Routing in Data Communication Networks, Part I: Principles", _E.E. Pub. No. 312_, Technion - Israel Inst. Technol., Sept. 1977.

[M3]   Moss, F.H. and Segall, A., "An Optimal Control Approach to Dynamic Rout-
       ing in Data Communication Networks, Part II: Geometrical Interpreta-
       tion", <u>E.E. Pub. No. 319</u>, Technion - Israel Inst. Technol., Jan. 1978.

[M4]   Mattheiss, T.H., "An Algorithm for Determining Irrelevant Constraints
       and all the Vertices in Systems of Linear Inequalities", <u>Opera-
       tions Research</u>, Vol. 21, No. 1, 247, Jan.-Feb. 1973.

[R1]   Rubin, D.C., "Vertex Generation and Cardinality Linear Programs", <u>Opera-
       tions Research</u>, 23, pp. 555-564, 1975.

[S1]   Segall, A., "The Modelling of Adaptive Routing in Data Communication
       Networks", <u>IEEE Trans. on Comm.</u>, COM-25, No. 1, pp. 85-95,
       January 1977.

ACKNOWLEDGEMENT