COMPUTATIONAL PRECEDENCE ORDERING IN MODULAR CASCADE SYSTEMS

A Thesis

Presented to

the Faculty of the Chemical Engineering Department

University of Houston

Houston, Texas.

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

by

Chilkunda K. Venkatesh

August, 1978

ACKNOWLEDGEMENT

My sincere and warmest thanks are due to Prof. Motard for both suggesting the topic and subsequent guidance. With great pleasure, I acknowledge the assistance rendered by my fellow graduate students, particularly Stig Wedel, for the many critical discussions. I also wish to thank the Department of Chemical Engineering for making financial support available through NSF Grant ENG 75 - 21544.

COMPUTATIONAL PRECEDENCE ORDERING IN MODULAR CASCADE SYSTEMS

An Abstract of a Thesis

Presented to

the Faculty of the Chemical Engineering Department

University of Houston

Houston, Texas.

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

by

Chilkunda K. Venkatesh

August, 1978

ABSTRACT

Situations where a non-redundant tear with respect to stream loops is impossible are frequently encountered in process simulation and give rise to difficulties in convergence of stream variables and overall heat and mass balances. Previous work has been directed at tearing the network at certain points rendering it acyclic and then precedence ordering the process units for computation. Such an approach is based on the structure of the directed graph alone and does not take into account in good measure the energy and mass flow patterns while precedence ordering. The idea in the present work has been to emphasise the stream loop as a major information recycle stream. Methods and criteria for precedence ordering simple and cyclic cascades have been established and shown to be better than the conventional cut set approach.

CONTENTS

CHAPTER I

INTRODUCTION

One approach to the computer aided design and optimization of a large chemical processing system is to develop an executive program coordinating the subroutines which perform the computations for the process units. Integral in this approach, is the specification of a precedence ordering which determines the sequence in which each unit subroutine is to be computed. An input process stream to a unit subroutine which has not been specified at the time of computation is called a recycle or torn stream, because, the stream must be cut by assuming initial values for all the process variables that are present in that stream. Later in the procedure, when the torn stream appears as an output from some unit subroutine, an iterative method is employed to force convergence of the torn stream i.e., the difference between the assumed and computed values of the stream variables, to within a specified tolerance.

A large body of theory has been developed and various criteria established to select these torn streams. The most important conclusion reached is that a non-redundant tear set has better convergence characteristics than those belonging to a redundant family (1). By redundancy we mean

that the members of the cut-set open the same recycle loop at more than one point. However, situations arise when such a non-redundant tear is not possible if recycle loops are to include non-simple cycles. Such systems where non-simple cycles called the stream loop exists, are more difficult to converge than normal problems, particularly, when we are considering overall mass balances.

Cascades are a particular class of examples which fall into this catagory. It is the purpose of this thesis to look at cascade systems, and develop methods to tackle the problem of precedence ordering and convergence in these situations.

CHAPTER II

GRAPH DEFINITIONS AND LITERATURE SURVEY

There are three main types of graphs - nondirected, directed graphs and graphs of a mixed type.

A nondirected graph consists of a number of nodes and a number of lines. Between the nodes and lines there exists an incidence relation which is defined as follows. Each line is incident either with one node or with two distinct nodes. Conversely, each node is incident with an arbitrary number of lines. This number may even be zero in which case the node is an isolated node.

Directed graphs are defined in the same way as nondirected graphs except for an additional requirement: each line must be oriented, i.e., for each line both a starting point and an end point must be specified, although the two may coincide.

In a mixed type of graph, some lines will be oriented and others not. Our interest is primarily in directed graphs and we will give more formal definitions.

Let N be an arbitrary set of nodes, and L an arbitrary set of lines and let N*N be the set of all unordered pairs of nodes in N. If $\alpha, \beta \in N$ and $\alpha \neq \beta$ then the pair formed by α and β is the set $\{\alpha, \beta\}$. If $\alpha = \beta$ then using the same notation we write $\{\alpha, \alpha\}$

Hence we can define

$$N*N = \{ \{\alpha,\beta\} \mid \alpha,\beta \in N\}$$

A non-directed graph (N,L,g) can be defined by N and L and a mapping $g:L \to N*N$, so that for each $b \in L$ and suitable $\alpha,\beta \in N$

$$gb = \{\alpha,\beta\}$$

A directed graph (N,L,f) can be defined by N and L and a mapping $f:L \to N \times N$, so that for each $b \in L$ and suitable $\alpha,\beta \in N$

$$fb = (\alpha,\beta)$$

Every directed graph (N,L,f) also defines a non-directed graph (N,L,g) where

$$gb = \{\alpha,\beta\} \text{ if } fb = (\alpha,\beta),\ b \in L, \alpha,\beta \in N$$

(N,L,g) is obtained from (N,L,f) by disregarding the order of the pairs of nodes, i.e., by disregarding the orientation of the lines.

A graph is called finite if N and L are finite sets. A subgraph G_1 of a graph G = (N,L,f) is defined by a graph (N_1,L_1,f_1) where $N_1 \subseteq N$, $L_1 \subseteq L$ and f_1 is induced by f which means that if $fb = (\alpha,\beta)$, $b \in L_1$ and $\alpha,\beta \in N_1$ then

$$f_1 b = (\alpha,\beta).$$

If $fb = (\alpha,\beta)$ we say that α is the starting point of b and β is its end point.

A path by definition is formed by a number of ordered nodes $\alpha_0,\ldots\ldots \alpha_n$ and ordered lines $b_1,\ldots. b_n$ where $n \geq 1$ such that $gb_i = \{\alpha_{i-1}, \alpha_i\}$, $i = 1,\ldots.n$.

This path is denoted by the sequence $(\alpha_0,b_1,\alpha_1,\ldots b_n,\alpha_n)$

from which we sometimes omit the nodes. We say that the path $(\alpha_0, b_1, \alpha_1, \ldots b_n, \alpha_n)$ connects α_0 and α_n, and that if it contains n lines, it consists of n steps or that its length is n.

A directed path is defined in the same way except that instead of requiring that $gb_i = \{ \alpha_{i-1}, \alpha_i \}$ we require that $fb_i = (\alpha_{i-1}, \alpha_i)$, $i = 1, \ldots n$. This directed path we again denote by $(\alpha_0, b_1, \ldots b_n, \alpha_n)$. A directed path is said to be directed from α_0 to α_n and α_0 and α_n are said to be the starting and ending points respectively.

A directed cycle is a directed path $(\alpha_0, b_1, \alpha_1, \ldots b_n, \alpha_0)$ where $\alpha_0 = \alpha_n$ and $n \geq 1$. The sequences $(\alpha_1, b_2, \alpha_2 \ldots b_n, \alpha_0, b_1, \alpha_1)$ and $(\alpha_0, b_1, \alpha_1, \ldots b_n, \alpha_0)$ are considered to represent the same cycle.

A simple directed path is a directed path all of whose nodes are distinct and a simple directed cycle is a directed cycle $(\alpha_0, b_1, \alpha_1, \ldots b_n, \alpha_0)$ where all the nodes $\alpha_0, \ldots \alpha_{n-1}$ are distinct.

A hinged directed cycle is a directed cycle in which only all the lines are distinct. This is also called a stream loop. A graph is connected if for every pair of nodes α and β there exists a path connecting α and β.

A cyclical loop is maximal if and only if it is cyclical and contains all other cyclical graphs as its subgraph. A maximal cyclical net contains no vertex of another larger net and hence for calculation purposes can be considered

separately. If the graph contains no simple loop it is called acyclic. The indegree of a node is the number of edges directed towards it, while the outdegree is the number of edges directed outwards from it. The sum of the indegree and outdegree is called the degree of the vertex.

Decomposition of nets

This consists of two parts

1. Identification of maximal cyclical nets.
2. Reduction of individual nets.

By the definition of a maximal cyclical graph, a recycle system must contain at least a simple loop. A common method of cutting this loop is to assume initial values for all the variables in any one of the streams which constitute the loop. The loop is said to be torn at the chosen point. If all the loops in the recycle system are torn in this manner, the resulting graph becomes acyclic and can then be precedence ordered to produce a set of new values for the torn variables. An iteration procedure is then performed to force the agreement between the assumed and computed torn variables to some preset tolerance. For a complex system, more than one such stream will have to be selected. A cut-set is said to be non-redundant if no simple loop is opened more than once by the cut streams. Upadhye and Grens (1) have shown that a non-redundant tear set has better

convergence characteristics than redundant tear sets. Three types of criteria are usually used to select the 'optimal' cut set (12)

1. To minimize the cut set of streams
2. To minimize the cut set of stream variables.
3. To minimize the largest eigenvalue of the sensitivity matrix related to the cut stream variables.

The problem of precedence ordering may now be divided into

- Identification of maximal nets
- Tearing
- Convergence

The main identification algorithms are shown in Table 2.1, and we can see that there are two major approaches : the path tracing methods (PTM) and powers of adjacency matrix methods (PAM). While PTM are difficult to program PAM has large core requirements. A summary of the major tearing algorithms is shown in Table 2.2.

From Tab. 2.2 we can summarize the basic approaches:

1. Steward's tearing algorithm
2. Integer programming technique
3. Boolean matrix operations
4. Branch and bound method
5. Boolean approach for bivalent optimization
6. Heuristic methods

Table 2.1 Identification of process flow networks

Author	Method
Norman (2)	PAM
Himmelblau (2,3)	PAM
Steward (4,5)	PTM
Sargent and Westerberg (6)	PTM
Christensen and Rudd (7)	PTM
Kehat and Shacham (8)	PAM
Ledet (9)	PAM
Jain and Eakmen (10)	PAM + PTM
Forder and Hutchinson (16)	PTM
Janicke and Biess (11)	PAM
Barkley and Motard (31)	PTM

Abbreviations: PAM powers of adjacency matrix

PTM path tracing methods

Table 2.2 Algorithms for tearing

Author	Method	Comments
Sargent and Westerberg (6)	Dynamic programming	Advantageous for a network with few units and many recycle loops
Crowe et al.(14)	Comparison of combinations	Not practical for large systems. Efficient for small number of nodes
Steward (5)	Loop tracing	Procedure results in one excess tear
Lee and Rudd (15)	Cyclic matrix operation	Advantageous for small systems and hand calculations
Forder and Hutchinson (16)	Cyclic matrix operation	Modification of Lee and Rudd procedure in an interactive mode
Lee, Christensen and Rudd (17)	Steward's procedure	Modification of Steward's procedure Inefficient for large systems
Christensen (18)	Bipartite graphs	Suitable for optimization and design calculations
Ledet and Himmelblau (9)	Loop tracing	

Tab. 2.2: Algorithms for tearing contd.

Author	Method	Comments
Westerberg and Edie (19)	Steward's procedure dynamic programming	Optimization of the output set to minimize the number of cut variables
Johns (20)	Search algorithm	Search algorithm arranges the nodes in a calculation order which creates a set of recycle nets of minimal sizes
Upadhye and Grens (21)	Dynamic programming	Efficient for large systems
Pho and Lapidus (22)	Graph approach	Graph simplification technique via repeated reduction of ineligible streams and two-way edges
Christensen and Rudd (7)	Graph approach	Minimum number of tears not gaurenteed
Ramirez and Vestal (23)	Elimination and structuring algorithm	Suitable for design calculation
Piehler (24,25)	Integer programming	Not practical for large systems
Jänicke and Bieß (11)	Occurence matrix operation	Iterative variables identified by inspecting a minimum number of rows and columns in the occurence matrix
Kevorkian and Snoek (26)		

Tab..2.2 Algorithms for tearing contd.

Author	Method	Comments
Garfinkel and Nemhauser (27)	Covering algorithm	Integer programming approach
Hammer (28)	BABO algorithm	Boolean approach for bivalent optimization
Wilde and Atherton (29)	Branch and bound solution	-
Barkely and Motard (31)	Graph apprcach	Signal flowgraph method. Minimal cut-set gaurenteed. Suitable for large systems

All the work in this field upto this point has been directed at finding the 'optimal' cut-set. Such an approach relies too heavily on the structure of the corresponding directed graph alone, and does not take into account the mass and energy flow patterns that exist in the system. This work has been directed at developing a feasible, but at the same time not completely heuristic, approach which can lay emphasis on this aspect as applied to cascade systems.

NOMENCLATURE FOR CHAPTER II

a_i, b_i	lines in the graph
f	mapping function for a directed graph
g	mapping function for a non-directed graph
L	set of lines in the graph
L_1	set of lines in the subgraph
N	set of nodes or vertices in the graph
N_1	set of nodes or vertices in the subgraph

Greek:

α, β	individual nodes or vertices

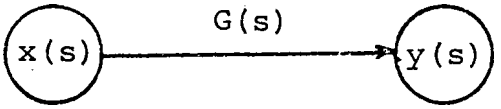
CHAPTER III

SIGNAL FLOW GRAPHS

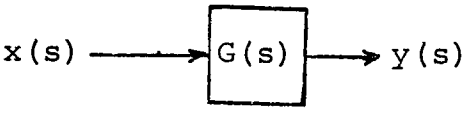
Signal flow graphs are a special type of directed graphs which provide the engineer with a method of analysing and solving a system described by a set of simultaeneous linear algebraic or differential equations without resorting to matrix calculations. The information contained in the signal flow graph is neither more nor less than that contained in the relation equations, but the signal flow graph does provide a visual representation of the system equations from which a logical reduction procedure can be effected. The visual representation of the system equations often makes the system more amenable to analysis.

Fig.3.1 illustrates the basic elements of a signal flow graph. The vertices (nodes) are the variables in the related equation connected by a directed line or branch, with the arrow pointing towards the dependent variable and away from the independent variable. Superimposed on the arrow is the symbol representing the branch transmittance (branch operator or branch gain) which represents the ratio of output to input, i.e., the ratio of the dependent to the independent variable. (A node may be a dependent variable in one part of the graph and an independent

Signal flow diagram



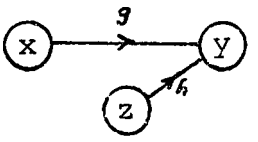
Block diagram



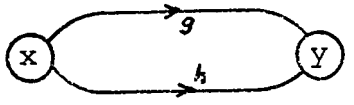
Equation $y(s) = G(s)\ x(s)$

Figure 3.1: Basic signal flow graph elements.

Addition

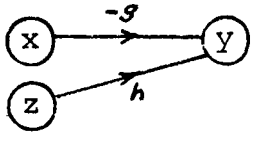


y=gx + hz

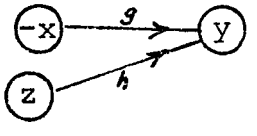


y=gx + hx

Subtraction

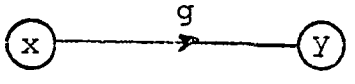


or



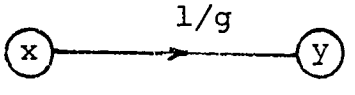
y=hz - gx

Multiplication



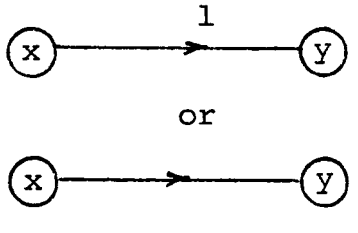
y = gx

Division



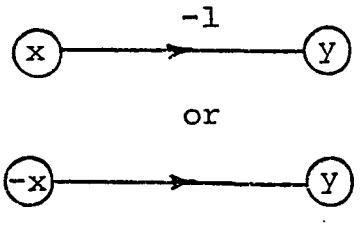
y = x/g

Identity or unit transmittance



y = x

Negative unit transmittance



y = -x

Figure 3.2: Basic rules for signal flow diagrams.

variable in another.) It corresponds to the transfer function if the variables are in Laplace transform space. A network of one or more branches is the 'signal flow graph'. The node represents both the operation of summation and the variables.

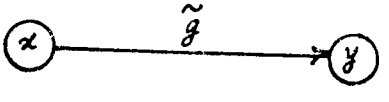
The rules for drawing signal flow graphs are as follows:

1. Material or information travel along the branch only in the direction of the arrow.
2. Any signal travelling along any branch is multiplied by the transmittance of that branch.
3. The value of the variable represented by any node is the sum of all inputs entering that node.
4. The value of the variable represented by any node is transmitted on all branches leaving that node.

The basic rules for addition, subtraction, multiplication and division are shown in Fig. 3.2. The rules for manipulating and consolidating signal flow graphs are as follows.

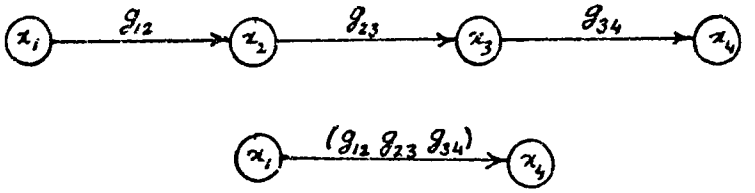
1. Addition rule:

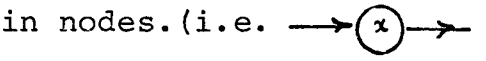
$$y = g_1 x + g_2 x + g_3 x$$
$$= (g_1 + g_2 + g_3) x$$
$$= \tilde{g} x$$



Parallel branches can be replaced by a single branch with a transmittance equal to the sum of the individual branch transmittances.

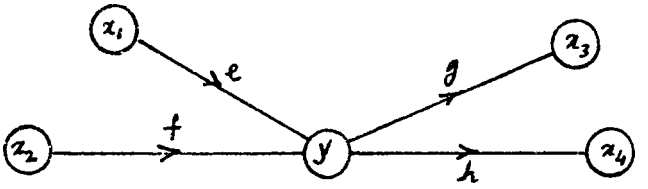
2. Multiplication rule:



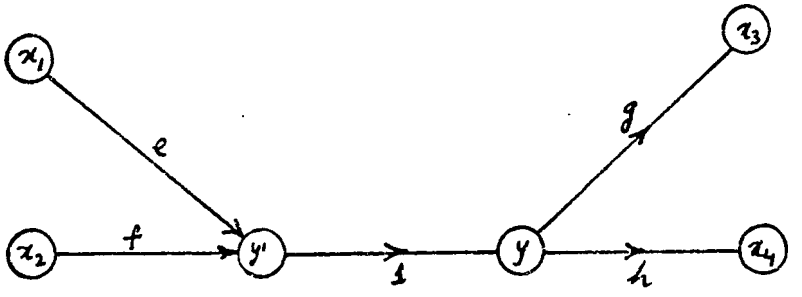
Series branches can be replaced by a single branch with transmittance equal to the product of the individual branch transmittances if all the intermediate nodes are chain nodes. (i.e.  is a chain node; a node containing a self loop is not a chain node)

3. Splitting:

A complicated node can be split up into a series of simpler ones by the use of unit transmittances.



is equivalent to



CHAPTER IV

SYSTEM SENSITIVITY AND SENSITIVITY MATRICES

System sensitivity, as a general concept, refers to the change in the output variable which can be attributed to a change in one of the system parameters (coefficients or in some cases system inputs). As a quantitative measure, sensitivity has value in allowing the engineer to predict possible changes in system outputs based on proposed or actual changes in system parameters. Sensitivity becomes especially important in recycle processes in which the possibility exists for the system output to influence itself. Systems with recycle have two sources of input: the normal flows into the system plus flows which depend upon the system output. The relative sensitivity of an iterative calculation can be defined as(35)

$$S = \left| \frac{x_i^{(n+1)} - x_i^{(n)}}{x_i^{(n)}} \right|$$

where the superscript refers to the cycle number of the iterative procedure. Briefly, the requirement for an iterative procedure to converge to a solution are as follows.

Firstly, the initial guess must be reasonably close. Second , the matrix of partial derivatives J, the Jacobian

must have moduli less than one.

For the set of system equations

$$f_1 (y_1, \ldots . y_n) = 0$$

$$\cdot$$

$$\cdot$$

$$f_n (y_1, \ldots . y_n) = 0$$

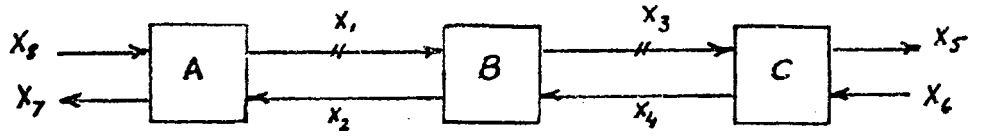
the matrix J is

$$J(y_1, \ldots y_n) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots\cdots\cdots & \frac{\partial f_1}{\partial y_n} \\ \cdot & & \\ \cdot & & \\ \cdot & & \\ \frac{\partial f_n}{\partial y_1} & \cdots\cdots\cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

where the J is evaluated at $(y_1, \ldots . y_n)$

We will now show how the Jacobian can be obtained for a linear system which has to be solved by iteration and that it is identical to the one obtained from the signal flow graph.

Let us suppose we have a three element cascade which looks like:



Let a_{ij} be the split fraction from stream i to j.
e.g., a_{12}, a_{42} imply the relation

$$X_2 = X_1 a_{12} + X_4 a_{42}$$

The sum of the a_{ij}'s leaving any node equals one due to mass balance requirements. Streams 1 and 3 will be chosen as cut streams. X_6 and X_8 are constant valued feed streams while X_5 and X_7 are product streams. We start the iterative procedure by assuming streams 1 and 3 at X_1^0 and X_3^0 while g_1 and g_3 are new estimates for streams 1 and 3 obtained after one cycle.

The computation sequence will be C → B → A.
The following relations are obtained:

$$X_4 = X_3^0 a_{34} + X_6 a_{64}$$

$$X_2 = X_1^0 a_{12} + X_4 a_{42} = X_1^0 a_{12} + X_3^0 a_{34} a_{42} + X_6 a_{64} a_{42}$$

And so

$$g_3 = X_3^1 = (X_3^0 a_{34} + X_6 a_{64}) a_{43} + X_1^0 a_{13}$$

and

$$g_1 = X_8\, a_{81} + X_2\, a_{21}$$

$$= X_8\, a_{81} + X_1^0\, a_{12}\, a_{21} + X_3^0\, a_{34}\, a_{42}\, a_{21} + X_6\, a_{64}\, a_{42}\, a_{21}$$

and hence

$$\frac{\partial g_1}{\partial X_1} = a_{12} a_{21}$$

$$\frac{\partial g_1}{\partial X_3} = a_{34}\, a_{42}\, a_{21}$$

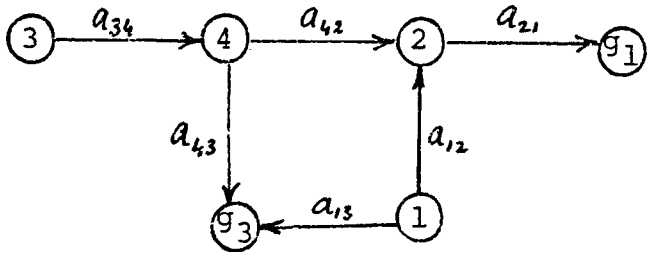
$$\frac{\partial g_3}{\partial X_1} = a_{13}$$

$$\frac{\partial g_3}{\partial X_3} = a_{34}\, a_{43}$$

Therefore

$$J = \begin{pmatrix} a_{12}\, a_{21} & a_{34}\, a_{42}\, a_{21} \\ a_{13} & a_{34}\, a_{43} \end{pmatrix}$$

Signal flow graph



Shown above is the signal flow graph for the system considered previously. The partial derivative $\partial g_1/\partial X_1$ is equal to the total transmittance from 1 to g_1 (12) calculated according to the principles and methods explained earlier.

Hence,

$$\partial g_1/\partial X_1 = a_{12}\ a_{21}$$

$$\partial g_1/\partial X_3 = a_{34}\ a_{42}\ a_{21}$$

$$\partial g_3/\partial X_1 = a_{13}$$

$$\partial g_3/\partial X_3 = a_{34}\ a_{43}$$

This is identical to the result obtained earlier.

The usefulness of sensitivity matrices arises in the prediction of the convergence rate of an iterative procedure. If we are sufficiently close to the solution that we can assume a linear approach to the solution, then the number of iterations required to reduce the error in the estimates by a factor ε is given by the following expression (33)

$$n = \frac{\log_{10} \varepsilon}{\log_{10} |\lambda_{max}|} \qquad [4.1]$$

where n is the number of iterations and λ_{max} is the largest eigenvalue of the sensitivity matrix. In the following chapter a general method to evaluate the sensitivity matrix for any arbitrary system will be derived.

CHAPTER V

A GENERAL METHOD FOR EVALUATING THE JACOBIAN

One of the techniques for reducing the number of iterations taken by a recycle system to converge is to minimize the largest eigenvalue of the Jacobian matrix. Upto this point, this has been done by comparing alternate cut sets. The approach here has been to achieve this by repetition of certain units in the cascade by following the longest stream loop. It is therefore necessary to have a method by which we can evaluate the Jacobian for any arbitrary system.

For the method developed here the following information is required:

a) A feasible cut set with reference to which the Jacobian is computed.

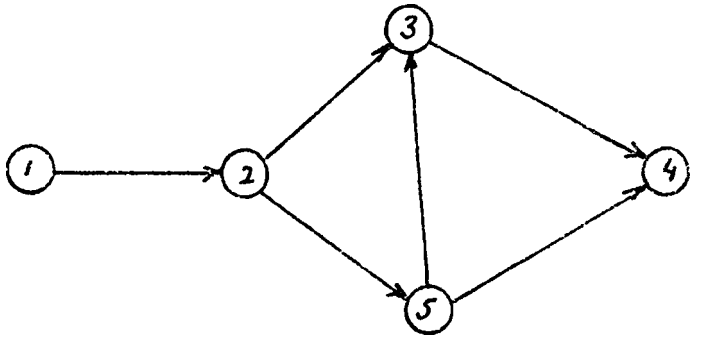
b) A precedence ordering

c) The split fractions at each unit.

The Jacobian is computed assuming one variable per stream viz., the total molar flow rate. However, this method can be extended to include individual components.

Associated with each digraph is a Boolean matrix R (associated matrix, relation matrix, transition matrix, adjacency matrix) which is a square matrix with as many rows (and columns) as the digraph has vertices. The element $r_{ij} = 1$ if there is a flow directed from vertex i to j, otherwise zero.

For example



R:

0	1	0	0	0
0	0	1	0	1
0	0	0	1	0
0	0	0	0	0
0	0	1	1	0

Figure 5.1: Directed graph and adjacency matrix

Note that the first column, fourth row and diagonal are all zero. This means that there there are no edges directed towards (1), no edges directed out of (4) and that there are no self loops.

By taking the n^{th} power of R, if r_{ij} is one then there exists a path n steps long from i to j in the graph. The powers of R are taken with the usual rules of matrix multiplication except that Boolean algebraic rules hold for individual elements.

viz., $x + y = \max(x,y)$

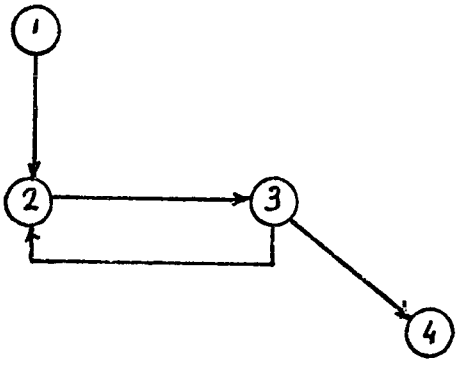
$x * y = \min(x,y)$

Another feature of the adjacency matrix associated with a directed graph is that it indicates when cyclical nets (dir.cycles) occur (35). If the graph has no directed cycles it is called acyclic and there will be some value N, corresponding to the longest path in the graph such that

$$R^{N+m} = 0 \quad \text{for all} \quad m \geq 1$$

For the example shown in Fig.5.1, N = 4, whereas

For the example shown below, no such N exists.



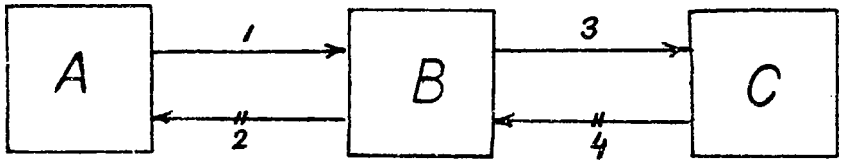
Now suppose that the rules for Boolean multiplication are substituted with that of regular multiplication and we are also given that the graph is acyclic. Then, the integers appearing as entries in R^n give the number of n-step paths from node i to j. For example in Fig.5.1,

R^2 =

$$\begin{matrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

i.e., the nodes (1,3), (1,5), (2,3), (5,4) are connected by two step paths, while there are two 2-step paths from node 2 to 4.

Now if the entries were replaced by the corresponding a_{ij}, the split fraction, the products appearing as entries will be the transmittances due to n-step paths. Thus by knowing the location corresponding to the Jacobian elements we can successively sum these to get the sensitivity matrix. This procedure is further guaranteed to terminate, since the signal flow graph obtained from the cut set must necessarily be acyclic. This procedure is illustrated by an example.



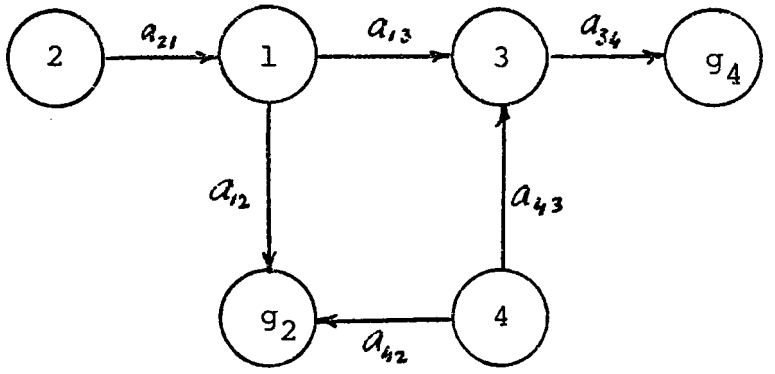


Figure 5.2: Example to illustrate the generalized method.

The adjacency matrix of the signal flow graph with a_{ij} entries is as follows:

	1	2	3	4	g_2	g_4
1	0	0	a_{13}	0	a_{12}	0
2	a_{21}	0	0	0	0	0
3	0	0	0	0	0	a_{34}
4	0	0	a_{43}	0	a_{42}	0
g_2	0	0	0	0	0	0
g_4	0	0	0	0	0	0

The underlined entries correspond to the Jacobian elements. For instance, if we need $\partial g_2/\partial X_4$, it is the total transmittance along all branches from X_4 to g_2. a_{42} corresponds to the transmittance along the one step path. Similarly by taking higher powers of R we can get the transmittance along the longer paths. These when summed together give the total transmittance from X_4 to g_2 which is equal to the corresponding partial derivative.

Shown below is R^2

	1	2	3	4	g_2	g_4
1	0	0	0	0	0	$a_{13}\ a_{34}$
2	0	0	$a_{21}a_{13}$	0	$\underline{a_{21}a_{12}}$	$\underline{0}$
3	0	0	0	0	0	0
4	0	0	0	0	$\underline{0}$	$\underline{a_{43}a_{34}}$
g_2	0	0	0	0	0	0
g_4	0	0	0	0	0	0

The contributions due to two-step paths have emerged.

R^3:

	1	2	3	4	g_2	g_4
1	0	0	0	0	0	0
2	0	0	0	0	$\underline{0}$	$\underline{a_{21}a_{13}a_{34}}$
3	0	0	0	0	0	0
4	0	0	0	0	$\underline{0}$	$\underline{0}$
g_2	0	0	0	0	0	0
g_4	0	0	0	0	0	0

$R^4 = 0$.

The following shows the summary of the entire operation.

JACOBIAN

From R^1

$$\begin{pmatrix} 0 & a_{42} \\ 0 & 0 \end{pmatrix}$$

From R^2

$$\begin{pmatrix} a_{21}a_{12} & a_{42} \\ 0 & a_{43}a_{34} \end{pmatrix}$$

From R^3

$$\begin{pmatrix} a_{21}a_{12} & a_{42} \\ a_{21}a_{13}a_{34} & a_{43}a_{34} \end{pmatrix}$$

There will be no further contributions from higher powers since $R^4 = 0$. This procedure has been programmed and the details are presented in the appendix.

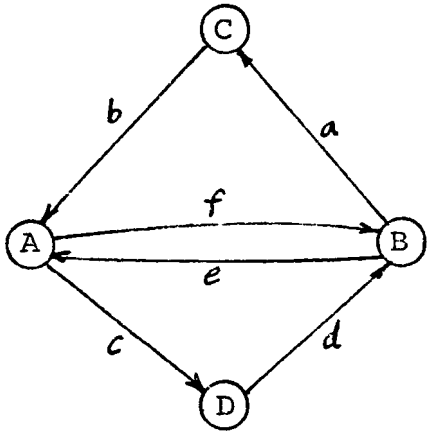
CHAPTER VI

REDUNDANT AND NON-REDUNDANT TEARING

In our earlier definition of a cyclic path a node and a stream could appear only once per cycle, and this was termed as a node loop. In such a case a non-redundant tear set can always be found with respect to the fundamental set of cycles in the graph(1). By a fundamental set, we mean that every cycle existing in the graph can be expressed as concatenations of parts of the cycles belonging to the fundamental set. For example, see Fig.6.1.

However, if we define stream loops as cyclic paths in which a node can be traversed more than once, but, every stream exactly once, then we frequently get a situation where a non-redundant tear with respect to stream loops is impossible. Fig.6.2 illustrates this situation. The prediction of the existence of stream loops is fairly straightforward. If we have a node which has at least two input edges and at least two output edges, then it means that the node can be traversed a second time, leaving by the remaining edge.

A cyclic path which includes every stream in the graph exactly once is called an Eulerian path. By



Fundamental cycles: 1. AfBaCbA

2. AcDdBeA

Other cycles:

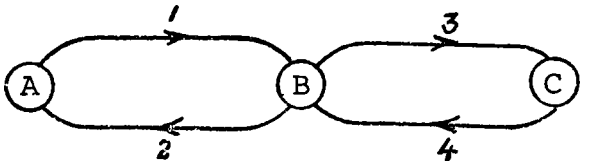
AcDdBaCbA = [AcDdB (2)] + [aCbA (1)]

AfBeA = [AfB (1)] + [eA (2)]

'+' means that the two strings are to concatenated.

( ) indicates which fundamental cycle the string has been extracted from.

Figure 6.1 : Fundamental cycles



Node loops: A 1 B 2 A

B 3 C 4 B

Cut sets: (1,3), (2,4), (1,4), or (2,3)

All equivalent by the Replacement rule (1).

Stream loop: A 1 B 3 C 4 B 2 A

No cut set can tear network without opening the stream loop at more than one point.

Figure 6.2: Non-redundant tearing

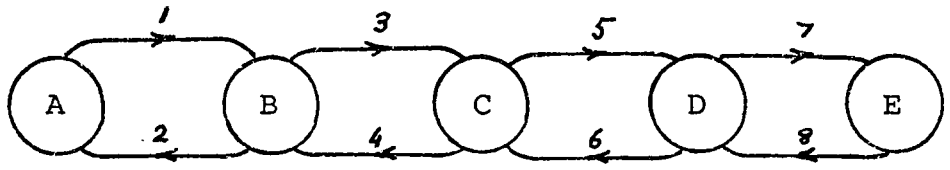
definition, it is obvious that an Eulerian cycle is also the longest cycle existing in the system. However, this need not be unique, and more than one Eulerian cycle can exist, as will be demonstrated later. It can be easily shown that the necessary and sufficient condition for the existence of an Eulerian cycle is that every node must have as many edges entering it, as are leaving it(13).

Cascades are a particular class of situations where the existence of the stream loop causes significant difficulties in convergence of both streams and overall mass balances. Cascades are frequently employed in the seperation of pure components from mixtures, occasionally accomplishing a chemical reaction. In each stage, two process streams are contacted and brought approximately to equilibrium with respect to each other. A number of such contacting stages are arranged in a cascade which produces the desired physical seperation or chemical change.

The number of such stages might vary anywhere from three or four in side strippers, to the order of a hundred or more in superfractionators. If such systems are solved by using the conventional cut set approach, every unit is repeated exactly once per cycle and so the major information feedback loop which exists due to the stream loop is not utilized.

Let us consider a five unit cascade and examine the sensitivity matrices obtained from the cut set and other

sequences. Hence we can estimate the eigenvalues and convergence rates.



a_{ij} = transmittance or split fraction from stream i to j

a_k = split associated with a particular module

e.g., $a_{21} = 1 - a_1$

$a_{12} = a_2$ and so on.

We have a four member cut set (2,4,6,8). (Or (1,3,5,7) or (1,4,5,8) all equivalent by the Replacement rule of Upadhye and Grens (1). For our purposes we will take (2,4,6,8)

(2),(4),(6),(8) are starting points for computation while g_2, g_4, g_6, g_8 are new estimates for the same streams after one iterative cycle. Shown on the next page is the signal flow graph for the sequence A→B→C→D→E, which arises from the cut set (2,4,6,8).

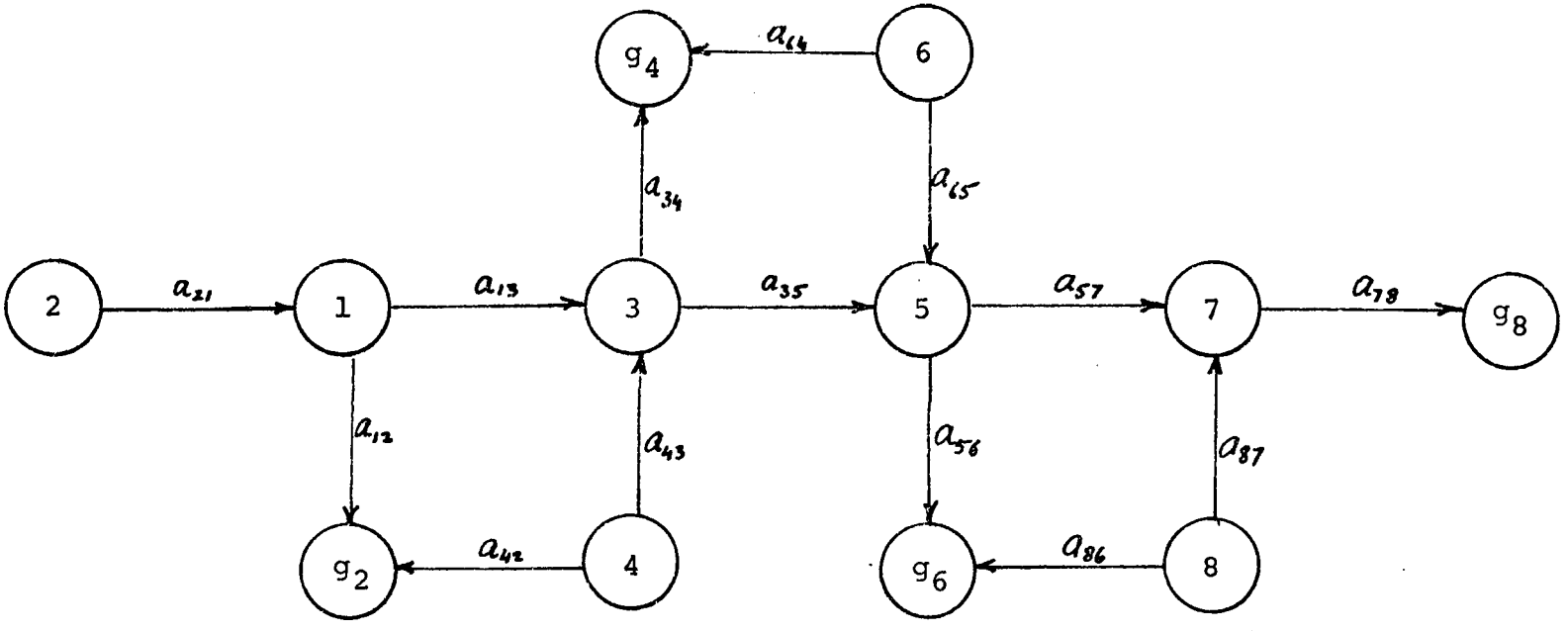


Figure 6.3 : Signal flow graph for sequence A→B→C→D→E

As explained previously, the Jacobain can be written as

$a_{21}a_{12}$	a_{42}	0	0
$a_{21}a_{13}a_{34}$	$a_{43}a_{34}$	a_{64}	0
$a_{21}a_{13}a_{35}a_{56}$	$a_{43}a_{35}a_{56}$	$a_{65}a_{56}$	a_{86}
$a_{21}a_{13}a_{35}a_{57}a_{81}$	$a_{43}a_{35}a_{57}a_{78}$	$a_{65}a_{57}a_{78}$	$a_{87}a_{78}$

The upper triangle of zeros gets correspondingly larger for bigger systems. This means that the partial derivative, for example $\partial g_2/\partial X_8 = 0$. Therefore any assumption or perturbation made to stream 8 will not affect stream 2 during that computation cycle. This introduces a delay into the system. The claim made here is that this delay is responsible for poor convergence and difficulties with overall mass balances.

Let us compare this with a different sequence. A smaller system with four units (A,B,C,D) will be used for illustrative purposes.

The sequence considered is 'ABABCBABCD'. The Jacobian for this will contain all non-zero elements. It

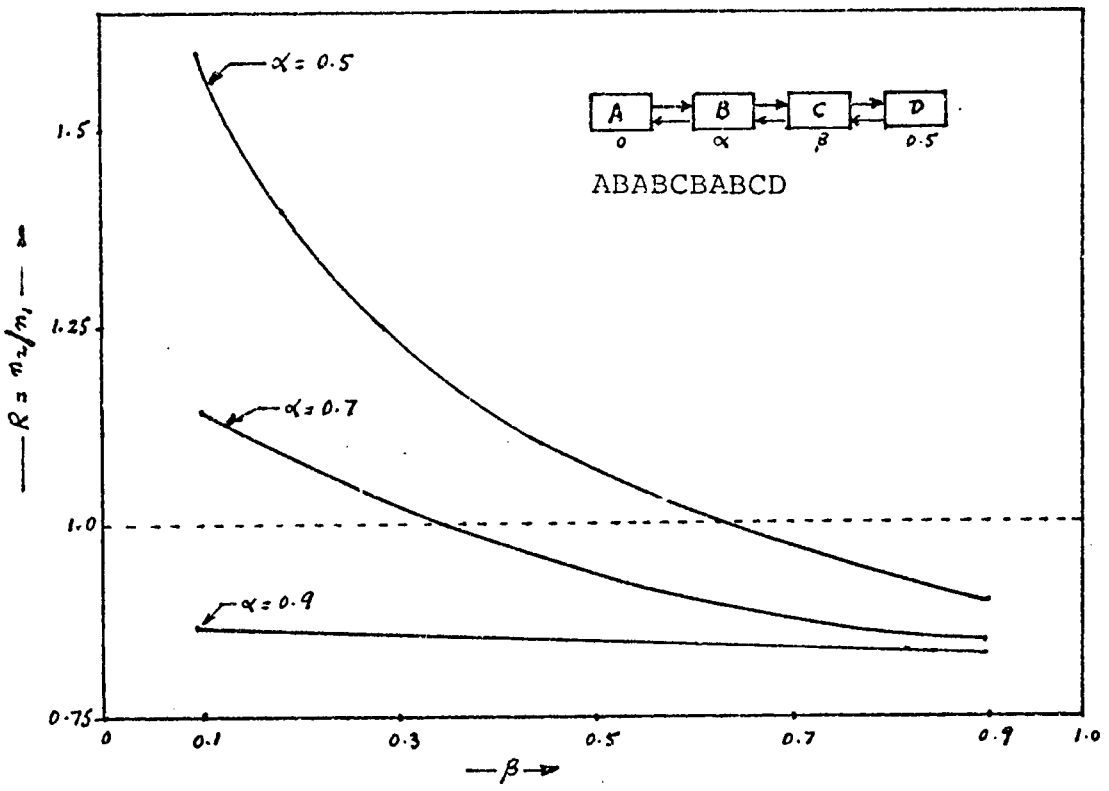


Figure 6.4: Comparison of sequences

is also less diagonally dominant than the Jacobian for 'ABCD'. This results in a lower maximum eigenvalue and a correspondingly lower number of predicted iterations. However, the computational effort expended per cycle for the longer sequence is correspondingly more. For comparing, the number of iterations predicted for both sequences will be weighted by the number of units in that cycle. The Jacobians and their eigenvalues were all evaluated using the generalized method developed in Chapter 5 .

Fig. 6.4 shows the result over a particular range of module split fractions. We can see that for particular ranges of split fractions, the longer sequence requires only 30% of the effort required to converge the cut-set sequence 'ABCD'. Similar results were obtained for larger systems.

This leads to the conclusion that we can improve convergence rate by minimizing the delay in the system, i.e., we should precedence order the streams in the cascade rather than the units. A binary distillation column and a thermally coupled distillation system were chosen for further examination and these are discussed in Chapters 7 and 8.

CHAPTER VII

LINEAR CASCADE - BINARY DISTILLATION COLUMN

The problem considered here is the binary distillation of benzene and toluene in a ten stage column. Stage 1 is a total condenser with a reflux ratio of 3.0 and stage 10 is a partial reboiler. Saturated liquid feed enters at stage 5.

Two cases are considered. First, when the top and bottom withdrawal rates are equal to half the feed flow rate irrespective of the feed composition and second, when the withdrawal rates are proportional to the amounts of benzene and toluene in the feed. The column was set up as a series of mixers and adiabatic flashes using the CHESS simulator (36). In this manner the sequence in which the stages were computed could be controlled. Tables 7.1 and 7.2 show the results.

The top entry is the number of iterations multiplied by the ratio (number of units in cycle)/(number of units in the cut-set cycle(10)). The second entry is the time ratio = (time taken for convergence by given sequence)/(time taken by the cut-set sequence). All simulations were run with a relative error tolerance of 1%.

Inspection of Table 7.1 reveals that all the proposed sequences perform better than the cut-set sequence as the

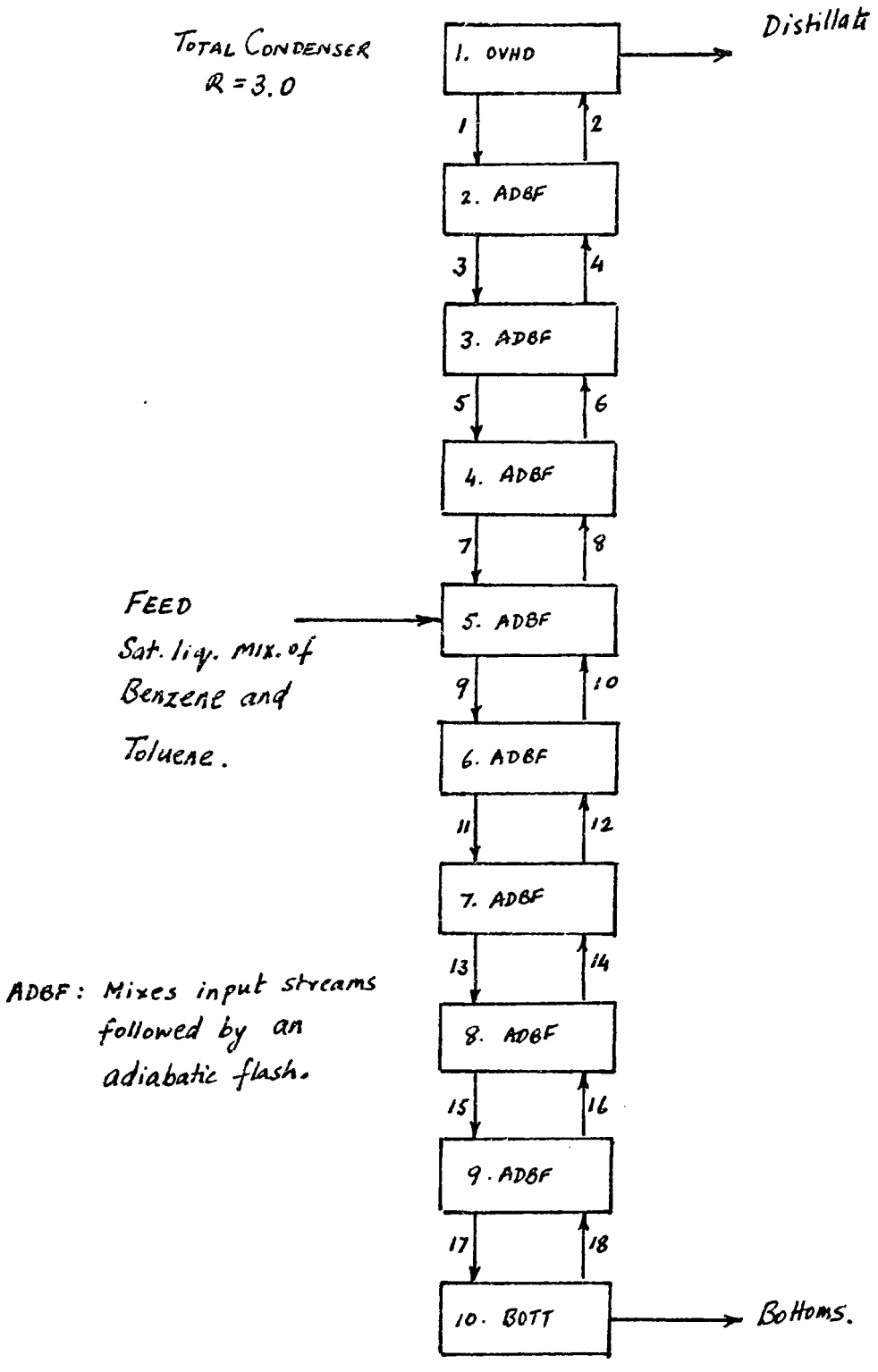


Figure 7.1 : Binary distillation column setup in CHESS simulator

B/T →	90/10	70/30	60/40	50/50
1→10	128 1.0	119 1.0	110 1.0	110 1.0
1→10→2	112 0.84	99 0.80	83 0.73	67 0.61
10→5→10 →1→9	173 1.06	92 0.77	56 0.50	48 0.44
1→5→1 →10→2	>169 >1.2	156 1.2	135 1.13	112 0.94
5→6→4 →10→1→4	141 1.03	128 1.01	106 0.91	84 0.72
5→7→3 →10→1→4	>169 >1.2	132 1.2	110 1.09	104 0.87

Binary distillation column on different sequences
Top and bottom flow rates same as B/T in feed.

Table 7.1

B/T→	90/10	80/20	70/30	60/40	50/50
1→10	110 1.0	110 1.0	110 1.0	110 1.0	110 1.0
1→10→2	27 0.29	31 0.30	32 0.32	32 0.32	67 0.61
10→5→10 →1→9	39 0.37	45 0.42	48 0.43	50 0.47	48 0.44
1→5→1 →10→2	36 0.33	36 0.33	39 0.36	47 0.39	112 0.94
5→6→4 →10→1→4	31 0.30	35 0.34	35 0.33	37 0.35	84 0.72
5→7→3 →10→1→4	36 0.33	39 0.36	39 0.36	44 0.44	104 0.87

Binary distillation column on different sequences 50-50 moles drawoff.

Table 7.2

Table 7.3: Convergence error data for 50-50 benzene toluene feed.

Sequence	Overall mass balance error %		
	Benzene	Toluene	Total molar flow
1→10 (cut-set)	0.94	3.43	1.25
1→10→2	2.7	2.5	0.12
10→5→10→1→9	1.43	2.01	0.29
1→5→1→10→2	2.39	2.21	0.09
5→6→4→10→1→4	2.48	2.27	0.10
5→7→3→10→1→4	1.93	1.05	0.08

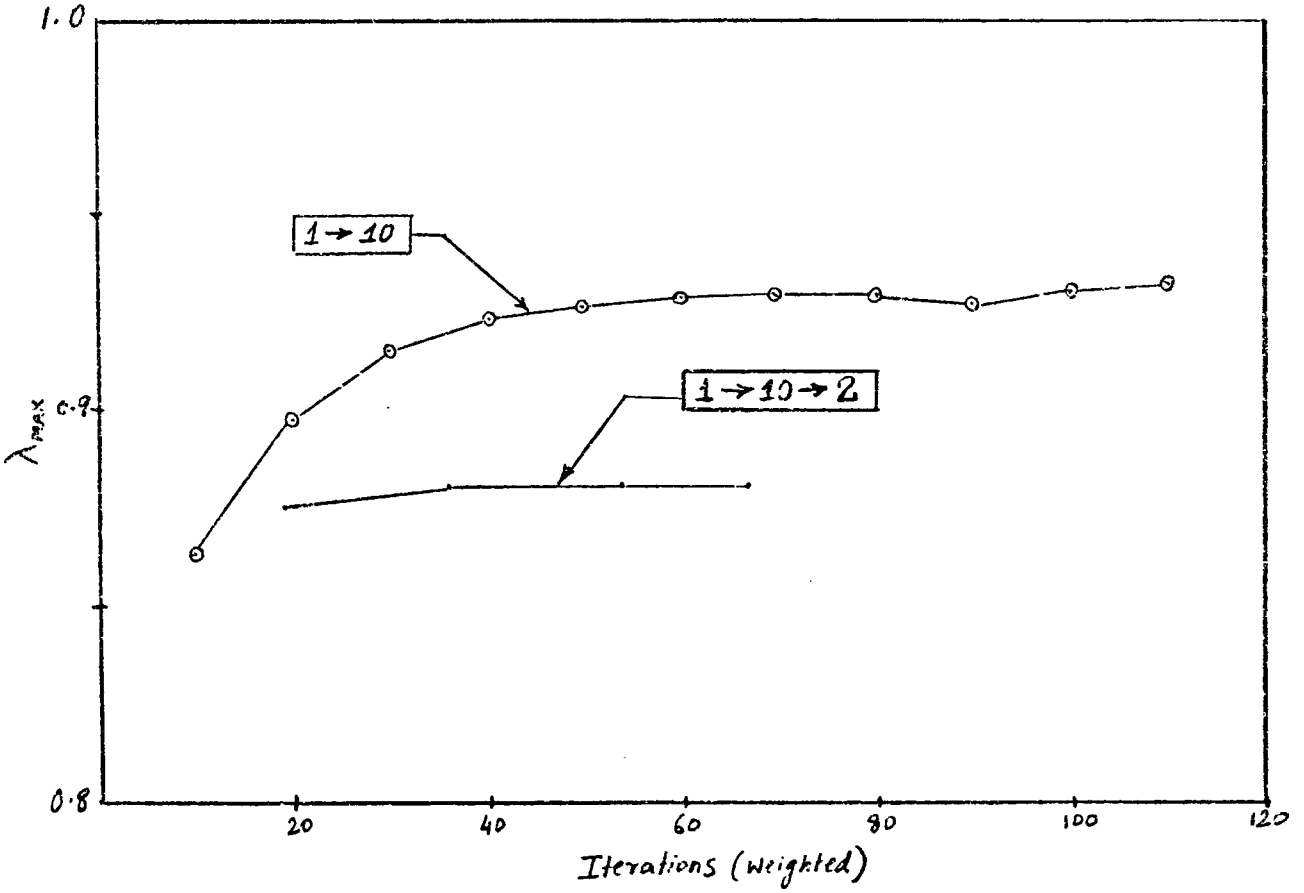


Figure 7.2: Eigenvalues of Jacobian with iteration.

benzene to toluene ratio becomes one. Sequence 10→5→10→1→9 performs much better than others over a smaller range, while 1→10→2 performs better than the cut-set in all cases.

Inspection of Table 7.2 reveals that all sequences perform better than the cut-set, while 1→10→2 is best in all except for equimolar mixture of benzene and toluene. Also the number of iterations taken take a sudden jump at this point. This is probably because the system now gets more strongly interactive. Attention is drawn to the fact that the sequence 1→10→2 performs better than the cut-set sequence more consistently than the others in this example.

Table 7.3 reveals an order of magnitude reduction in the overall mass balance error when based on total flow rates. Though individual component errors are lower than the cut-set error in most cases, they are still above the stream tolerances. This means that the overall mass flow pattern has stabilized much faster than the individual components. The individual errors for the components are observed to be opposite in sign, accounting for the low overall error. This suggests a compensating convergence acceleration to be performed, since we now have a bound for the total flow rates. By compensating, we mean that if the convergence acceleration of a component increases its flow rate, the others need not be accelerated independently, but can be proportionately reduced to satisfy

the total mass flow rate.

Figure 7.2 shows the variation in the eigenvalues of the Jacobian for the cut-set sequence and the sequence 1 10 2 with the number of iterations. The longer sequence has been multiplied by 1.8 to compensate for its extra length. This is reasonable in this case since most of the units take approximately the same computation time. The Jacobian has been based on the overall flow rate and the eigenvalues computed using the program and method developed in Chapter 5. The use of overall flow rate is justified in retrospective . Using equation 4.1, we get

$$n = \log(0.01)/\log(0.8815)$$
$$= 36.51$$

and when multiplied by 1.8 to account for its length we get $n' = 66$. The actual number of iterations taken works out to 67.

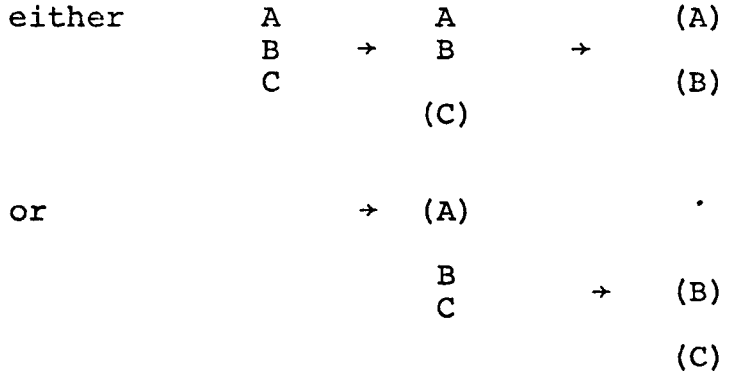
All said, we can conclude that the sequence based on including every stream exactly once can be expected to perform better than the cut-set generated sequence. This principle is extended to a more complex case of a thermally connected distillation system and is discussed in the next chapter.

CHAPTER VIII

CYCLIC CASCADE - THERMALLY COUPLED DISTILLATION SYSTEM

A distillation system contains a thermal coupling when a heat flux is utilized for more than one fractionation, and the heat transfer between fractionation sections occurs by direct contact of vapor and liquid. Compared with a conventional system, thermally coupled distillation systems can seperate close boiling components with considerable saving of heating and cooling costs (32). The separation of a multicomponent mixture is conventionally accomplished in a series of columns numbering one less than the number of products, each having a condenser and a reboiler.

In a ternary mixture of A,B, and C, in a conventional scheme we can have



In a thermally coupled system, initial seperation is made between A and C while (A,B) are seperated in the top

of the second column and (B,C) at the bottom. The separations are essentially binary and can be carried out without interference from the third component. Details of the case are shown in Figure 8.1. The primary purpose was to simulate a cyclic cascade system and no attempt was made to design the columns for a sharp seperation.

A result observed in the binary column case will be used here viz., a sequence of units based on evaluating each stream exactly once is likely to be better than one based on cut-sets. This means that we have to find the Eulerian paths in the system. This was done by considering the reduced system shown in Figure 8.2 .

Finding all the stream loops in a directed graph is itself a fairly difficult task. There are again two major approaches: path tracing and powers of adjacency matrix methods. Path tracing methods are believed to be more efficient from the standpoint of running time in spite of relatively large storage requirements (34). The algorithm presented by Weinblatt(30) was programmed in PL/1 with some minor modifications. Details of this program are given in the appendix.

There are a total of sixty one loops in Fig.8.2 out of which twelve are Eulerian. These are shown in Table 8.1.

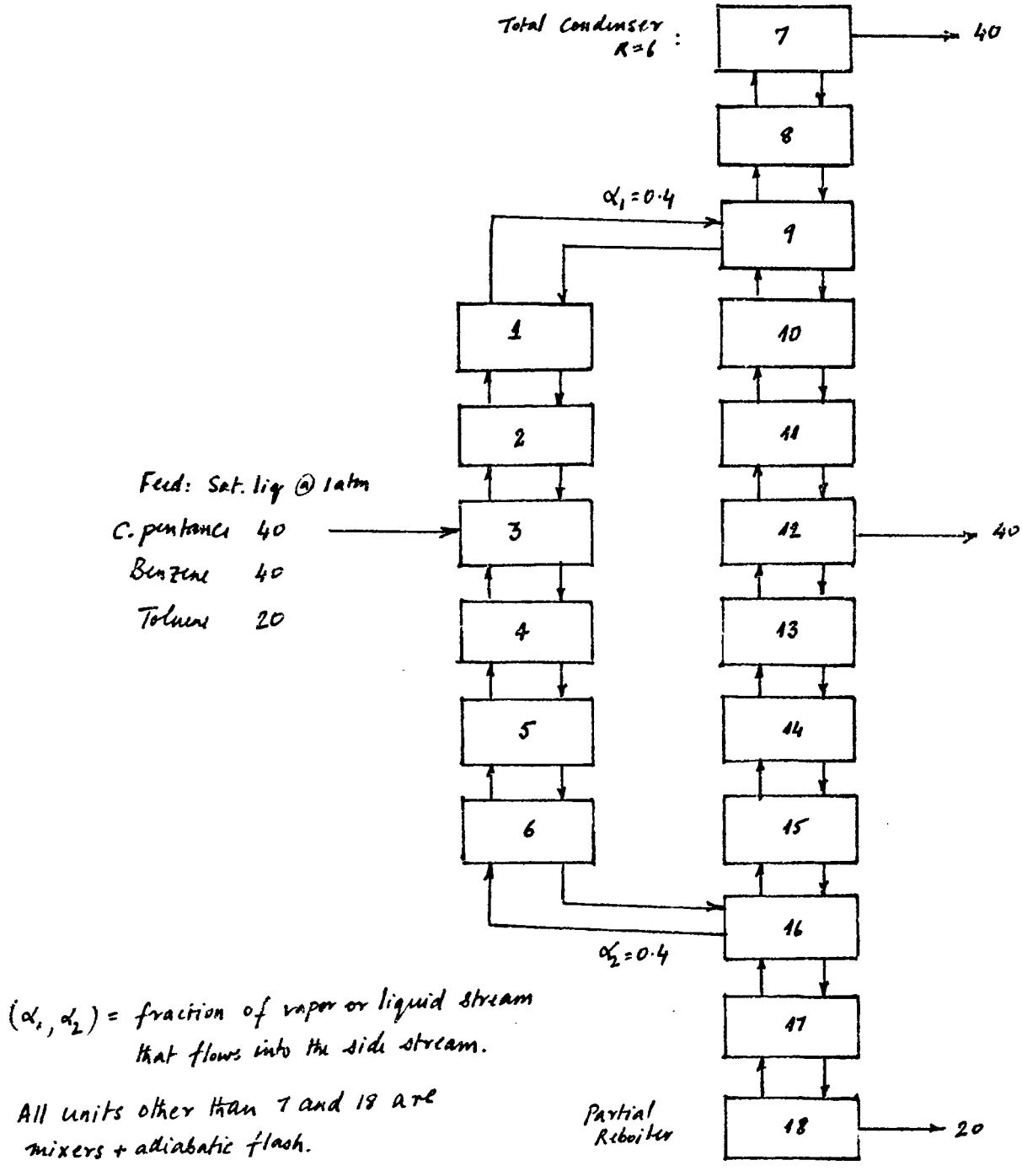


Figure 8.1: CHESS setup for thermally coupled system.

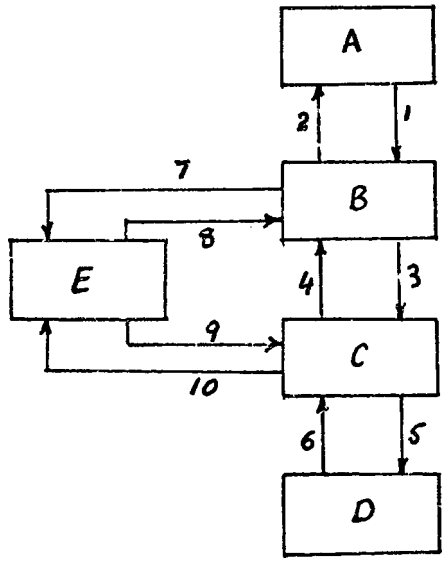


Figure 8.2 Reduced structure

Total number of stream loops = 61

Total number of Eulerian paths = 12

No.	Stream loop										
1.	1	3	4	7	9	5	6	10	8	2	1
2.	1	3	10	8	7	9	5	6	4	2	1
3.	1	3	10	9	5	6	4	7	8	2	1
4.	1	3	5	6	4	7	9	10	8	2	1
5.	1	3	5	6	10	8	7	9	4	2	1
6.	1	3	5	6	10	9	4	7	8	2	1
7.	1	7	8	3	10	9	5	6	4	2	1
8.	1	7	8	3	5	6	10	9	4	2	1
9.	1	7	9	4	3	5	6	10	8	2	1
10.	1	7	9	5	6	4	3	10	8	2	1
11.	1	7	9	5	6	10	8	3	4	2	1
12.	1	7	9	10	8	3	5	6	4	2	1

Table 8.1 Eulerian loops in Fig 8.2 .
Loops are listed by stream number sequence

Table 8.2: Sequences arising from Eulerian paths

No.	Sequence
1.	A B C B E C D C E B
2.	A B C E B E C D C B
3.	A B C E C D C B E B
4.	A B C D C B E C E B
5.	A B C D C E B E C B
6.	A B C D C E C B E B
7.	A B E B C E C D C B
8.	A B E B C D C E C B
9.	A B E C B C D C E B
10.	A B E C D C B C E B
11.	A B E C D C E B C B
12.	A B E C E B C D C B

Table 8.3: Iteration and simulation time data on cyclic cascades

Iterations taken by cut-set: 130

Sequence #	Iterations*	Time ratio
1	102	0.77
2	102	0.77
3a**	100	0.75
3b	104	0.77
3c	125	0.92
4	100	0.75
5	104	0.78
6a	100	0.76
6b	102	0.77
6c	128	0.94
7a	102	0.77
7b	100	0.76
7c	128	0.94
8a	100	0.75
8b	102	0.76
8c	123	0.91
9	106	0.80
10	104	0.79
11	100	0.76
12	102	0.77

* : Multiplied by the ratio of the number of units

** : See text for explanation of 'a','b',and 'c'.

Table 8.4: Convergence data on cyclic cascades

Sequence Reference Number	Overall mass balance error %			
	C.Pentane	Benzene	Toluene	Total Flow
Cut-set	0.72	6.34	5.56	3.94
3c	2.53	0.80	5.17	0.30
5	3.11	0.04	4.89	0.25
6b	3.37	0.19	5.16	0.24
7a	3.27	0.14	5.15	0.22
9	3.02	0.22	4.67	0.19

Table 8.2 shows the sequences which arise by following the Eulerian paths. The basic sequences are interpreted as follows:(see Fig.8.1 also)

Sequence # 1: A B C B E C D C E B leads to a unit precedence ordering

7→8→9→10→11→12→13→14→15→14→13→12→11→10→9→1→2→3→4→5→6→15→16→17→18→17→16→15→6→5→4→3→2→1→9→8

Sometimes more than one interpretation is possible. (Seq. 3,6,7,and8) For example sequence # 3 is A B C E C D C B E B could be interpreted as

3a: E passed in both directions at first occurence.
3b: E passed in both directions at second occurence
3c: E passed in both directions at both occurences.

The result of the simulations on all these possibilities are shown in Table 8.3. The results are fairly conspicuous. All the sequences perform better than the cut-set. Two catagories arise; one with a time ratio between 0.75 and 0.8 and the other with the time ratio over 0.90. The one with the greater ratio has unit E repeated in both directions at both occurences. We can safely conclude that in cases where more than one interpretation is possible, any one which covers the units in both directions once will suffice.

Table 8.4 shows convergence data on some of the

sequences. Similar trends as in the binary distillation case are observed. The overall mass balance based on total flow rates is cut by an order of magnitude, while some others are reduced significantly. This is again due to the fact that the overall flow rates are stabilizing much faster than the individual components. As suggested earlier in Chapter 7, this can be made the basis for a compensating acceleration algorithm.

It is now established quite clearly that in case of cascade systems, faster convergence is obtained by following the longest stream loop, and that this invariably performs better than the cut-set approach.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATION FOR FURTHER WORK

The importance of the stream loop as a major information recycle stream has been established. A general method to evaluate the sensitivity matrix for any arbitrary system has been developed. Cut-set approaches to solving cascade systems have been shown to have a delay inherent in them by considering their sensitivity matrices. It was further shown that this was reponsible for poor convergence characteristics. Stream loop sequences stabilize much faster at lower eigenvalues than cut-set sequences. It has been shown that following an Eulerian path in a cascade is definitely superior to cut-set approaches.

This work has established a framework for a new generation of precedence ordering algorithms which can be designed to handle very large systems more effectively. This is possible by developing a method to recognize an embedded cascade structure from a global viewpoint and being able to partition the graph as such. We have observed that overall mass balances based on total flow rates converge much faster than the individual components. This can form the basis for a different type of convergence acceleration algorithm which can utilize the bounds established.

REFERENCES

1. Upadhye R.S. and Grens E.A.,AIChE J.,21, 1,136(1975)
2. Norman R.L.,AIChE J.,11,450 (1965)
3. Himmelblau D.M. (Ed.):Decomposition of Large Scale Problems (Nort-Holland Publ.Comp. 1973)
4. Steward D.V.,SIAM Rev.,4,321 (1972)
5. Steward D.V.,SIAM Num. Anal.,2, 2,345(1965)
6. Sargent R.W.H. and Westerberg A.W.,Trans.IChE,42,T190 (1964)
7. Christensen J.H. and Rudd D.F.,AIChE J.,15,94(1969)
8. Kehat E. and Shacham M.,Process Tech.Inter.,18,1/2, (1973);18,4/5(1973)
9. Ledet W.P. and Himmelblau D.M.,Adv.Chem.Eng.,8 , 186 (Academic Press 1970)
10. Jain Y.V.S. and Eakmen J.M.,Chem.Eng.Comp.Workshop, Vol 2,No.W5
11. Janicke W. and Beiss G.,Chem. Techn.,26,740 (1974)
12. Genna P.L. and Motard R.L.,AIChE J.,21,417,(1975)
13. Ponstein J.,"Matrices in Graph and Network Theory", Van Gorcum and Comp. (1966)
14. Crowe C.M. et.al.,"Chemical Plant Simulation", (Prentice Hall 1971)

15. Lee W. and Rudd D.F.,AIChE J.,12,1184 (1966)

16. Forder G.J. and Hutchinson W.P.,Chem.Eng.Sci., 24,771 (1969)

17. Lee W., Christensen J.H. and Rudd D.F.,AIChE J., 12,1104 (1966)

18. Christensen J.H., AIChE J., 16, 177 (1970)

19. Westerberg A.W. and Eddie F.C.,Chem.Eng.J.,2,9 (1971)

20. Johns W.R.,"Mathematical Considerations in Preparing General Purpose Computer Programs for the Design or Simulation of Chemical Processes",Paper presented at the EFCE Conference (Apr. 1970,Florence)

21. Upadhye R.S. and Grens E.A.,AIChE J.,18,533 (1972)

22. Pho T.K. and Lapidus L.,AIChE J.,19,1170 (1973)

23. Ramirez W.F. and Vestal C.R.,Chem.Eng.Sci.,27,2243 (1972)

24. Piehler J.,Math.Of.und Statistik,3,83 (1972)

25. Ponstein J.,J.Soc.Indust.Appl.Math.,9,2,233 (1961)

26. Kevorkian A.K. and Snoek J.,"Decompositions of Large Scale Systems" in Ref.(3)

27. Garfinkel L. and Nemhauser G.L.,"Optimal Set Covering: a Survey" in 'Perspective on Optimization'(A.M.Geoffrion Ed.,Addison Wesley,Reading,Mass. 1972)

28. Hammer P.L.,"A Boolean Approach for Bivalent Optimization" in 'Optimization and Design',Avriel M.,Rijckaert M.J. and Wilde D.J. (Eds.) (Prentice Hall 1973)

29. Wilde D.J. and Atherton L.F.,"Branch and Bound Solution of Combinatorial Problems in Design", in Ref.(28)

30. Weinblatt H.,ACM J.,19,43 (1972)

31. Barkely R.W. and Motard R.L.,Chem.Eng.J.,3,265,(1972)

32. Stupin W.J. and Lockhart F.J.,Chem.Eng.Prog.,68, No.10,71 (1972)

33. Barchers D.E.,Ph.D. Thesis,Oregon State University(1975)

34. Allen F.E.,Program Optimization Research Report RC-1959 IBM Watson Research Center,Yorktown Heights,N.Y.(Apr.1966)

35. Himmelblau D.M.,'Process Analysis and Simulation', John Wiley and Sons.(1968)

36. Motard R.L. and Lee H.M.,CHESS User's Guide, Univ. of Houston. (1971)

APPENDIX A

In Chapter 5 the basis for a generalized Jacobian generator was established. This appendix briefly covers the programming details, information input and output.

We need four basic sets of information:

1. The structure of the process flow network
2. The cut-set with reference to which the Jacobian is evaluated.
3. The proposed precedence ordering
4. The split fractions between streams at each node.

The example shown in Figure A1 will be used for illustration. It is assumed here that

a) The streams are numbered 1 through $N_{streams}$

b) The nodes are numbered 1 through N_{nodes}

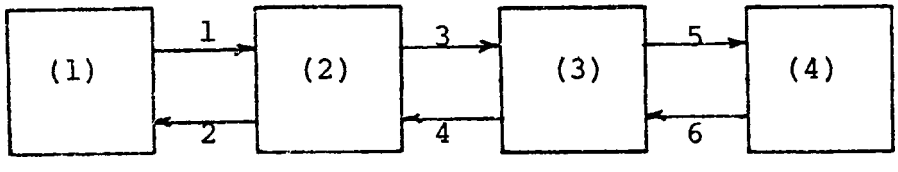


Figure A1.

The structure is coded by means of the following vectors as follows:

NFTCIN — This specifies all possible stream to stream connections existing in the system. For example 1 to 3 and 1 to 2 arise from stream 1 and node 2. This is coded in a field of length six as follows

Entry= (From stream #)*1000 + To stream #

For the above example NFTCIN would be

/1002,1003,2001,3004,3005,4002,4003,5006,
6004,6005/

NPROC — This specifies the stream numbers coming into and leaving each node from 1 through N_{nodes}. For the above example NPROC would be

/2,1,1,4,2,3,3,6,4,5,5,6/

NSTIOV — This specifies the indegree and outdegree of the nodes from 1 through N_{nodes}. For the above example NSTIOV would be

/1001,2002,2002,1001/

The entries are again coded as

Entry = (Indegree)*1000 + Outdegree

Only reduced network is to be considered; feed and product streams are to be eliminated

NSEQ	The precedence ordering of the nodes is stored in this vector.
NCSET	Contains cut-set. The streams in the cut-set must be specified in the same order it is required in the precedence ordering. For example if we have 1→2→3→4 as the precedence ordering then NCSET = /2,4,6/, while if we have 4→3→2→1 then NCSET = /5,3,1/.
SFIN	This specifies the split fractions between streams at each node. The order must exactly follow the stream connections specified in the NFTCIN vector.
NFT	Signal flow vector;contains SFG in vector storage
NDNFT	Length of NFT
NSTRMS	Total number of streams
NUNITS	Total number of nodes or units
NSU	Stream update vector; used while creating signal flow graph; length must be specified NSTRMS
NB,NC,SFA SFB,SFC	Vectors used in matrix multiplication; Adequate length depending on the problem must be specified.
NCODL	Vector contains locations of split fractions in SFIN to be associated with NFT entries.

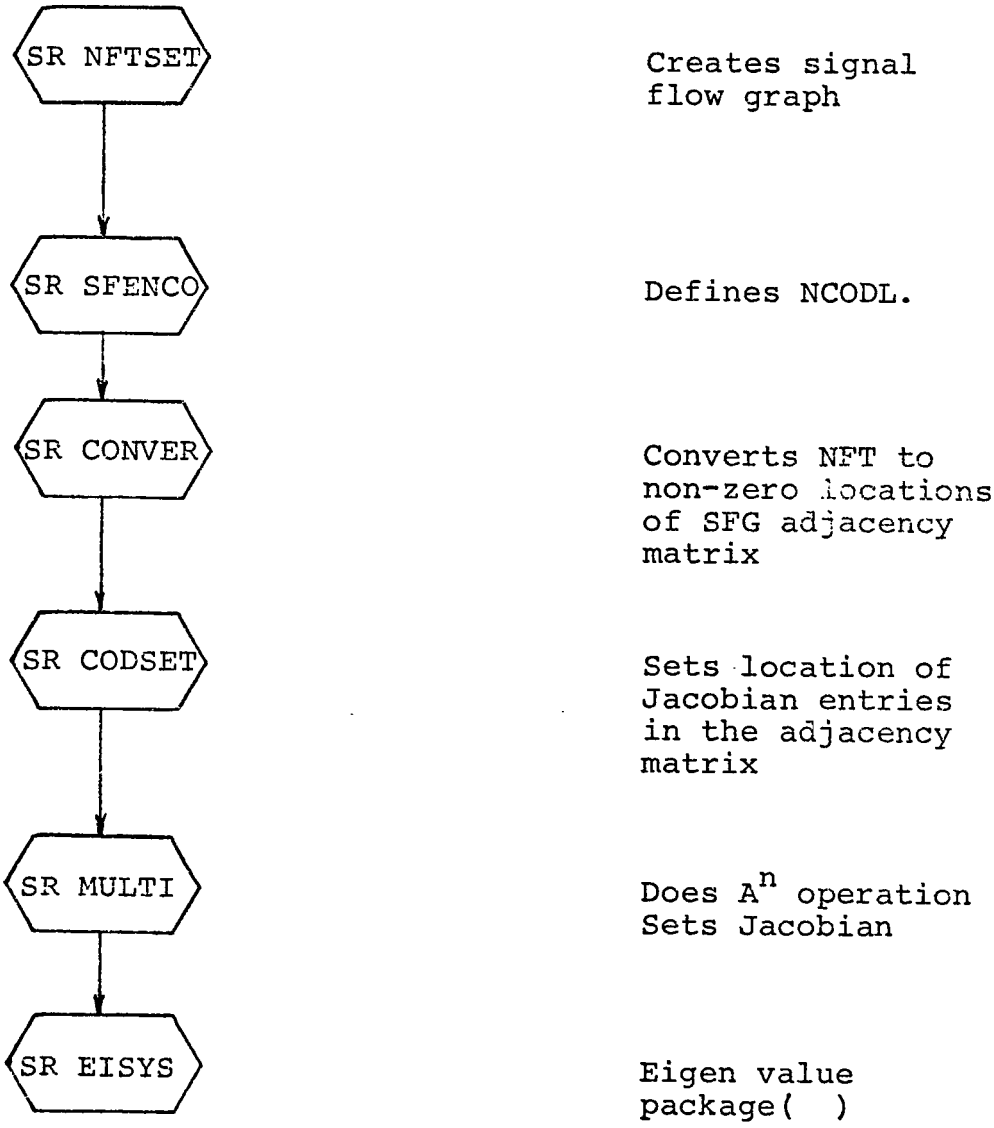


Figure A2: Overall orgainization of subroutines in generalized Jacobian generator.

```
      EXTERNAL NXY,FFF
      DOUBLE PRECISION EPS
      INTEGER NVF1(5)/4H(3X,,2H  ,4H(2X,,4HF11.,4H8)) /
      INTEGER NDIGIT(10)/1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,2H10/
      INTEGER NVF2(9)/4H(//,,3H1X,,2H  ,3HI1,,2H  ,4H(1X,,4HF11.,
     & 4H6),/,2H/)/
      DIMENSION NFT2(150),NPROC(150),NB(150),NC(150),SFA(150),SFB(150),
     1 SFC(150),NCODL2(150)
      DIMENSION NFTCIN(10),SFIN(10),NSTIOV(4),NSU2(6)
      DIMENSION NACJ2(3),NCSET(3),NC2(3)
      DOUBLE PRECISION XJ2(3,3)
      DIMENSION NSEQ2(6)
      NU2=6
      DATA NFTCIN/10,2,11,3,2001,3004,3005,4002,4003,5006,5004,6005/
      DATA NPROC/2,1,1,4,2,3,3,6,4,5,5,6,138*0/
      DATA NSTIOV/1001,2*2002,1001/
      DATA NCSET/2,4,5/
      DATA LEN1,NDNFT,NCUT,NSTRM,NUNITS/10,150,3,5,4/
      DATA SFIN/10*.5/
C READ IN SEQUENCES TO BE EXAMINED
      READ(5,640)(NSEQ2(KLM),KLM=1,NU2)
      NVF1(2)=NDIGIT(NCUT)
      NVF2(3)=NDIGIT(NCUT)
      NVF2(5)=NDIGIT(NCUT)
C
      CALL DRIVE(NFT2,NPROC,NSU2,NSEQ2,NU2,NSTRM,NSTIOV,NUNITS,LAST2,
     &           NCODL2,NFTCIN,LEN1,NACJ2,NCSET,NCUT,NDNFT)
      WRITE(6,666)(NSEQ2(KL2),KL2=1,NU2)
      CALL MULTI(NFT2,NDNFT,LAST2,SFIN,XJ2,NCUT,NACJ2,NSTRM,NSU2,NCODL2
     1 ,LEN1,NB,NC,SFA,SFB,SFC)
      WRITE(6,NVF1)((XJ2(L9,L1),L1=1,NCUT),L9=1,NCUT)
      DO 224 KK=1,NCUT
224   NC2(KK)=0
      INDEX=-1
      ND=NCUT
      NCOL=NCUT
      EPS=1.D-06
      CALL EISYS(ND,NCOL,NCUT,INDEX,EPS,NC2,XJ2)
      WRITE(6,NVF2)(NC2(K1),K1=1,NCUT),(XJ2(K2,1),K2=1,NCUT)
5001  CONTINUE
5000  CONTINUE
640     FORMAT(40I2)
666   FORMAT(//,'  SEQ UNDER CONSIDERATION ',25I4,//)
      STOP
      END
      SUBROUTINE MULTI(NA,NDNFT,LAST,SFIN,XJ,NCUT,NACJ,NSTRM,NSU,NCODL,
```

```
     1 LEN1,NB,NC,SFA,SFB,SFC                                          )
       DIMENSION NACJ(NCUT),NA(NDNFT),NB(NDNFT),NC(NDNFT),SFA(NDNFT),
     1 SFB(NDNFT),SFC(NDNFT),NSU(NSTRM),NCODL(NDNFT),SFIN(LEN1)
       DOUBLE PRECISION XJ(NCUT,NCUT)
       INTEGER NDIGIT(10)/1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,2H10/
       INTEGER NVF1(5)/4H(3X,,2H   ,4H(2X,,4HF11.,4H8)) /
       NVF1(2)=NDIGIT(NCJT)
C  INITIALIZE  NB TO NA AND NC TO ZERO AND SFA TO SFB
C  INITIALIZE ALL VARIABLES TO ZERO
       DO 299 I=1,NDNFT
       NB(I)=0
       NC(I)=0
       SFA(I)=0
       SFB(I)=0
       SFC(I)=0
299    CONTINUE
C TO GET DIMENSION OF MATRICES NA,NB,NC SUM UP NSU
       NDIMA=0
       DO 100 I=1,NSTRM
100    NDIMA=NDIMA+NSU(I)
       DO 3001 M=1,NCUT
       DO 3002 MM=1,NCUT
       XJ(M,MM)=0.
3002   CONTINUE
3001   CONTINUE
       DO 980 IX=1,LAST
       NB(IX)=NA(IX)
       LLL=NCODL(IX)
       SFA(IX)=SFIN(LLL)
       SFB(IX)=SFA(IX)
980    CONTINUE
C  CHEX FOR ONE STEP PATHS AND ACC.
       DO 710 I=1,NCUT
       DO 720 J=1,NCUT
       N1=FLOAT(NACJ(I))/1000.
       N2=FLOAT(NACJ(J))/1000.
       N3=NACJ(J)-N2*1000
       LOC=N1*1000 + N3
       XJ(I,J)=XJ(I,J) + FFF(LOC,NA,SFA,NDNFT,LAST)
720    CONTINUE
710    CONTINUE
       WRITE(6,NVF1)((XJ(IZ1,IZ2),IZ2=1,NCUT),IZ1=1,NCUT)
       KOUNT=0
9999   LOC=0
       KOUNT=KOUNT+1
       IF(KOUNT .EQ. 1) NLASTB=LAST
C  ESTABLISH THE CO-ORDINATE OF NC VIA (I,J)
```

```
      DO 200 I=1,NDIMA
      DO 300 J=1,NDIMA
      T=0.
      DO 400 K=1,NDIMA
C  ENCODE LOCATION OF ELEMENS OF NA AND NB RESPY.
      L1=100*I+K
      L2=100*K+J
C  SEARCH SECOND LIST SINCE IT WILL PROGRESSIVELY HAVE LESSER ELEMENTS
      T2=FFF(L2,NB,SFB,NDNFT,NLASTB)
      IF (T2) 5,6,5
5     T1=FFF(L1,NA,SFA,NDNFT,LAST)
      T=T+T1*T2
6     CONTINUE
400   CONTINUE
      IF(T) 7,8,7
C  ENCODE LOCATION IN NC
7     LOCAC=100*I+J
      LC=LC+1
      NC(LC)=LOCAC
      SFC(LC)=T
8     CONTINUE
300   CONTINUE
200   CONTINUE
      NLASTB=LC
      IF(LC .EQ. 0) RETURN
237   FORMAT(I4,2X,3(I1),E15.8))
C NOW FILL OUT ELEMENTS OF XJ VIA NACJ,NC,SFC
      DO 510 I=1,NCUT
      DO 520 J=1,NCUT
      N1=FLOAT(NACJ(I))/1000.
      N2=FLOAT(NACJ(J))/1000.
      N3=NACJ(J) - N2*1000
      LOC=N1*1000 + N3
      XJ(I,J)=XJ(I,J) + FFF(LOC,NC,SFC,NDNFT,LC)
520   CONTINUE
510   CONTINUE
      WRITE(6,NVF1)((XJ(IZ1,IZ2),IZ2=1,NCUT),IZ1=1,NCUT)
      WRITE(6,2000)
2000  FORMAT(///)
C  SET NB TO NC AND ERASE NC
      DO 610 JX=1,NDNFT
      NB(JX)=NC(JX)
      NC(JX)=0
      SFB(JX)=SFC(JX)
      SFC(JX)=0.
610   CONTINUE
      GO TO 9999
```

```
      RETURN
      END
      SUBROUTINE NETSET(NET,NPROC,NSU,NSEQ,NU,NSTRM,NSTIOV,NUNITS,LAST)
      DIMENSION NET(150),NPROC(150)
      DIMENSION NSU(NSTRM),NSEQ(NU),NSTIOV(NUNITS)
C     PURPOSE:  TO ESTABLISH SIGNAL FLOW INFORMATION STORED IN NET VECTOR
CC              FOR ANY GIVEN COMPUTATION SEQUENCE
C
      LAST=1
      DO 110 I=1,NU
C
CCC   NU       NUMBER OF UNITS IN THE SEQUENCE (PER CYCLE)
CCC   NUC      UNIT NUMBER UNDER CONSIDERATION
C
      NUC=NSEQ(I)
      CALL LOCATE(NSTIOV,NPROC,NUC,NINU,NOLU,NLOC,NUNITS)
C     NINU     NUMBER OF STREAMS ENTERING UNIT
CCC   NOLU     NUMBER OF STREAMS LEAVING UNIT
CCC   NLOC     STARTING LOCATION OF INFO. IN NPROC VECTOR
CCC   NFROM    INPUT STREAM UNDER CONSIDERATION
CCC   NTO    OUTPUT STREAM UNDER CONSIDERATION
C
      NLOCO=NLOC+NINU+NOLU -1
      NLOCIN=NLOC+NINU-1
      NLOCI1=NLOCIN+1
C     UPDATE CYCLE BEGINS
CC    FROM LOOP WILL UPDATE NSU IF IT IS ZERO
C
      DO 200 J1=NLOC,NLOCIN
      NFROM=NPROC(J1)
      IF(NSU(NFROM).EQ.0)NSU(NFROM)=1
200   CONTINUE
C     TO - LOOP WILL UPDATE NSU WHENEVER STREAM IS COMPUTED
      DO 210 J2=NLOCI1,NLOCO
      NTO=NPROC(J2)
      NSU(NTO)=NSU(NTO)+1
210   CONTINUE
C     ENCODE STREAM NUMBERS
      DO 220 J1=NLOC,NLOCIN
      NFROM=NPROC(J1)
      NETFR=1000*NSU(NFROM) + NFROM
      DO 230 J2=NLOCI1,NLOCO
      NTO=NPROC(J2)
      NETTO=1000*NSU(NTO)+NTO
      NET(LAST)=NETFR*10000+NETTO
      LAST =LAST+1
230   CONTINUE
```

```
220     CONTINUE
110     CONTINUE
C  LAST WILL CONTAIN THE LOCATION OF THE LAST NON-ZERO ELEMENT OF NFT---
        LAST=LAST-1
        RETURN
        END
        SUBROUTINE LOCATE(NSTIOV,NPROC,NUC,NINU,NOLU,NLOC,NUNITS)
        DIMENSION NPROC(15 )
        DIMENSION NSTIOV(NUNITS)
C
        NUC1=NUC-1
        NTEMP=0
        IF (NUC1) 300,300,200
  200   DO 100 I=1,NUC1
        N1=FLOAT(NSTIOV(I))/1000.
        N2=NSTIOV(I)-N1*1000
        NTEMP=NTEMP+N1+N2
100     CONTINUE
  300   NLOC=NTEMP+1
        NINU=FLOAT(NSTIOV(NUC))/1000.
        NOLU=NSTIOV(NUC)-NINU*1000
        RETURN
        END
        SUBROUTINE CONVER (NFT,LAST1,NSU,NSTRMS)
        DIMENSION NFT(150)
        DIMENSION NSU(NSTRMS)
C  THIS CONVERTS THE ELEMENTS OF THE NFT VECTOR INTO THE COORDINATE
C   LOCATIONS OF THE SIGNAL FLOW MATRIX
        DO 100 I=1,LAST1
        NFROM=FLOAT(NFT(I))/1000.
        NTO=NFT(I)-1000*NFROM
        NROW=NXY(NFROM,NSU,NSTRMS)
        NCOL=NXY(NTO,NSU,NSTRMS)
        NFT(I)=NROW*1000+NCOL
C  NOTE THAT THE NFT FIELD HAS BEEN REDUCED TO 2*3=6
100     CONTINUE
        RETURN
        END
        INTEGER FUNCTION NXY(NENTRY,NSU,NSTR)
        DIMENSION NSU(NSTR)
        NRECUR=FLOAT(NENTRY)/1000.
        NSN=NENTRY - NRECUR*1000
        IF(NSN .EQ. 1) GO TO 200
        NSN1=NSN - 1
        NTOT=0
        DO 100 J=1,NSN1
100     NTOT=NTOT+NSU(J)
```

```
      NXY=NTOT+NRECUR
      RETURN
200   NXY=NRECUR
      RETURN
      END
      SUBROUTINE CODSET(NACJ,NCSET,NCUT,NSU,NSTRM)
      DIMENSION NACJ(NCUT),NSU(NSTRM),NCSET(NCUT)
C    THIS DETERMINES WHICH ELEMENTS OF THE SIGNAL FLOW MATRIX SHOULD
C   BE SUMMED TO GET THE ELEMENTS OF THE JACOBIAN
C    NSTRMC = CUT STRM UNDER CONSIDERATION
C KOUNT KEEPS TRACK OF LAST ENTRY(NE ZERO) OF NACJ IN CASE CUT STRM =1
C IS NOT SPECIFIED FIRST
      KOUNT=0
      DO 100 I=1,NCUT
      NSTRMC=NCSET(I)
      IF(NSTRMC .EQ. 1) GO TO 101
      NTOT=0
      NSTRM1=NSTRMC - 1
      DO 200 J=1,NSTRM1
200   NTOT=NTOT+NSU(J)
      NROW=NTOT+1
      NCOL=NTOT+NSU(NSTRMC)
      NACJ(I)=NROW*1000 + NCOL
      KOUNT=KOUNT+1
      GO TO 100
101   NROW=1
      NCOL=NSU(1)
      KOUNT1=KOUNT+1
      NACJ(KOUNT1)=NROW*1000 + NCOL
100   CONTINUE
      RETURN
      END
      SUBROUTINE SEENCO(NFT,LAST,NCODL,NFTCIN,LEN1      )
      DIMENSION NFT(150),NCODL(150)
      DIMENSION NFTCIN(LEN1)
      DO 100 I=1,LAST
C ENTRY TO BE DECODED RECOVERED FROM NFT=NDE
      NT1=FLOAT(NFT(I))/10000.
      NT2=NFT(I)-10000*NT1
      NT3=FLOAT(NT1)/100.
      NT4=FLOAT(NT2)/100.
      NT5=NT1-100.*NT3
      NT6=NT2-100.*NT4
      NDE=100.*NT5+NT6
C   SEARCH IN NFTCIN(LEN1) VECTOR
      DO 200 J=1,LEN1
      IF(NFTCIN(J) .EQ. NDE) GO TO 300
```

```
      GO TO 200
300   NCODL(I)=J
      GO TO 400
200   CONTINUE
400   CONTINUE
100   CONTINUE
      RETURN
      END
      REAL FUNCTION FFF(L,NX,SFX,NDIM,NLAST)
      DIMENSION NX(NDIM),SFX(NDIM)
C PURPOSE: TO DETERMINE IF 'L' EXISTS ON LIST NX
      DO 100 I=1,NLAST
      IF(L-NX(I)) 6,7,6
7     FFF=SFX(I)
      RETURN
6     CONTINUE
100   CONTINUE
      FFF=0.
      RETURN
      END
      SUBROUTINE DRIVE(NFT,NPROC,NSJ,NSEQ,NU,NSTRM,NSTIOV,NUNITS,LAST,
     &                 NCODL,NFTCIN,LEN1,NACJ,NCSET,NCUT,NDNFT      )
      DIMENSION NFT(NDNFT),NPROC(NDNFT),NSU(NSTRM),NSEQ(NU),NSTIOV(
     &          NUNITS),NCODL(NDNFT),NFTCIN(LEN1),NACJ(NCUT),NCSET(NCUT
     &)
      DO 110 I=1,NDNFT
      NFT(I)=0
      NCODL(I)=0
110   CONTINUE
      DO 101 I=1,NSTRM
101   NSU(I)=0
      DO 102 I=1,NCUT
102   NACJ(I)=0
      CALL NFTSET(NFT,NPROC,NSU,NSEQ,NU,NSTRM,NSTIOV,NUNITS,LAST)
      WRITE(6,100)(NFT(I),I=1,LAST)
      CALL SFENCO(NFT,LAST,NCODL,NFTCIN,LEN1)
      WRITE(6,100)(NCODL(JJ),JJ=1,LAST)
      CALL CONVER (NFT,LAST,NSU,NSTRM)
      WRITE(6,100)(NFT(I),I=1,LAST)
      WRITE(6,100)(NSU(J),J=1,NSTRM)
      CALL CODSET(NACJ,NCSET,NCUT,NSJ,NSTRM)
      WRITE(6,100)(NACJ(J),J=1,NCUT)
100   FORMAT(3X,11(1,/)
      RETURN
      END
```

SEQ UNDER CONSIDERATION 1 2 3 2 3 4

0.0 0.0 0.0

0.0 0.0 0.0

0.0 0.0 0.0

0.25000000 0.0 0.0

0.0 0.0 0.0

0.25000000 0.50000000 0.25000000

0.25000000 0.12500000 0.0

0.12500000 0.0 0.0

0.25000000 0.62500000 0.25000000

0.31250000 0.12500000 0.06250000

0.12500000 0.62500000 0.0

0.25000000 0.62500000 0.31250000

0.31250000 0.15625000 0.06250000

0.12500000 0.62500000 0.03125000

0.25000000 0.62500000 0.31250000

0.31250000 0.15625000 0.07812500

0.12500000 0.62500000 0.03125000

0.25000000 0.62500000 0.31250000

0.31250000 0.15625000 0.07812500

0.12500000 0.62500000 0.03125000

0.25000000 0.62500000 0.31250000

000 0.54384763 0.14365237 0.00000000 ← EIGENVALUES.

APPENDIX B

The algorithm programmed has been presented by Weinblatt (30). To find the stream loops we input the signal flow graph of the given directed graph. Since the nodes of the original graph become the streams and the streams, nodes, we can find all the stream loops in the original graph by finding all the node loops in its signal flow graph.

The inputs are number of nodes, number of streams, followed by XREF ($N_{streams}$,4) which is entered as follows:

XREF(i,1) = stream number

XREF(i,2) = starting node number

XREF(i,3) = terminal node number of that stream

XREF(i,4) = zero; this space is used during program execution.

All other dimensions are allocated during execution time.

```
(STRG,STRZ,SUBRG):
CYCLES:PROCEDURE OPTIONS(MAIN);
DECLARE (NODES,NSTRMS)FIXED DECIMAL(2,0);
GET LIST(NODES,NSTRMS);
M1:BEGIN;/* ARRAY AREA FOR OTHER VARIABLES ASSIGNED HERE*/
DECLARE (XREF(NSTRMS,4),STATV(NODES),RECUR,NSTR,CYCN,K,I,
        J,K1,K2,NODE)FIXED DECIMAL(2,0);
DECLARE (TEMP0(3*(NODES+NSTRMS)) FIXED DECIMAL(2,0) CONTROLLED;
DECLARE LOOPCY(CYCN,NSTRMS)FIXED DECIMAL(1,0) CONTROLLED,T2 CHARACTER
(2),XX FIXED DECIMAL(2,0);
DECLARE DEBUG BIT(1);  DEBUG='1'B;
 DEBUG='0'B;

        DECLARE (INDEG(NODES),OUTDEG(NODES))FIXED DECIMAL(2,0)
                                                CONTROLLED;


DECLARE(CYC(125))CHARACTER(3*(NODES+NSTR))VARYING CONTROLLED;
        DECLARE (LABEL1,LABEL2,LABEL3,LABEL4) LABEL,
                (VERT,LASTARC) FIXED DECIMAL(2,0);
        DECLARE TT CHARACTER(3*(NODES+NSTR))VARYING CONTROLLED;
        ALLOCATE INDEG,OUTDEG;
ON ERROR PUT LIST (CYC);
 PUT LIST('  XREF - INPUT')      ;
GET LIST(XREF)COPY; PUT LIST(' NOW CALLING DEGREE');
 CALL DEGREE;
  PUT LIST('  NODE#  INDEGREE  OUTDEGREE  STATV')SKIP(2);
DO K=1 TO NODES;PUT EDIT(K,INDEG(K),OUTDEG(K),STATV(K))(SKIP,X(3),F(2)
,X(7),F(2),X(9),F(2)   ,X(8),F(2));END;
 NSTR=NSTRMS;
  PUT LIST('  NODE#  INDEGREE  OUTDEGREE  STATV')SKIP(2);
DO K=1 TO NODES;PUT EDIT(K,INDEG(K),OUTDEG(K),STATV(K))(SKIP,X(3),F(2)
,X(7),F(2),X(9),F(2)   ,X(9),F(2));END;
PUT LIST('  STRM#  SV   TV  STATUS')SKIP(2);DO K1=1 TO NSTRMS;DO K2=1
TO 4;PUT EDIT(XREF(K1,K2))(X(3),F(2),X(4),F(2),X(3),F(2),X(4),F(2));
END;END;
  CYCN=1;
        FREE INDEG,OUTDEG;
 ALLOCATE TT,CYC;
  CYC(*)='';
L100:   CALL SELECT;                            GO TO LABEL1;
L200:   CALL EXTEND;                            GO TO LABEL2;
L300:   CALL BACKUP;                            GO TO LABEL3;
LEXAM :CALL EXAMINE;                            GO TO LABEL4;
L400:   CALL ADDCYCL;                           GO TO L300;
L500:   RECUR=0;
```

```
  CALL CONCAT('N'||ENCODE(VERT));     GO TO L300;
DEGREE:PROCEDURE;
DECLARE(SV,TV)FIXED DECIMAL(2,0);
STATV=0;
 INDEG,OUTDEG=0;
L1:DO I=1 TO NSTRMS BY 1; SV=XREF(I,2);TV=XREF(I,3);
IF((SV=0)|(TV=0))THEN GO TO L2; ELSE;OUTDEG(SV)=OUTDEG(SV)+1;
INDEG(TV)=INDEG(TV)+1;L2:END L1; END DEGREE;
SELECT:PROCEDURE;
L1:     DO I=1 TO NODES;
        IF(STATV(I)=0)THEN DO;
                                VERT=I;
                                STATV(I)=1;
                                TT='N' || ENCODE(VERT);
                                LABEL1=L200;
    PUT SKIP LIST(' NODE SELECTED ',VERT);
                                RETURN;
                              END;
ELSE;  END L1;  LABEL1=LSTOP;
 PUT LIST(' NO MORE ELIGIBLE NODES --PROGRAM TERMINATED');
END SELECT;


EXTEND:PROCEDURE;

 DECLARE XL FIXED DECIMAL(2,0);
        VERT=DECODE(SUBSTR(TT,LENGTH(TT)-1,2));
  IF (DEBUG=1) THEN DO; PUT SKIP;
PUT LIST(' ENTERING EXTEND VERT=',VERT,'NSTR=',NSTR);
PUT LIST('  STRM#   SV    TV  STATUS')SKIP(2);DO K1=1 TO NSTRMS;DO K2=1
TO 4;PUT EDIT(XREF(K1,K2))(X(3),F(2),X(4),F(2),X(3),F(2),X(4),F(2));
END;END;
PUT LIST(' STATV=',STATV);
 END;ELSE;
L1:     DO I=1 TO NSTR;
        IF(XREF(I,2)=VERT) THEN
                           L2:DO;
                                IF(XREF(I,4)=0)THEN
                           L3:DO;
                                LASTARC=XREF(I,1);
                                TT=TT||('S'||ENCODE(LASTARC));
                                LABEL2=LEXAM;
                                       XREF(I,4)=2;
  IF (DEBUG=1) THEN DO; PUT SKIP;
PUT SKIP LIST('LASTARC=',LASTARC,'TT=',TT);
 END;ELSE;
```

```
                                        RETURN;
                                    END L3; ELSE;
                                END L2; ELSE;
            END L1;
  XL=LENGTH(TT);
 IF(XL=3)THEN DO;TT='';  GO TO LLL;END;
 ELSE  TT=SUBSTR(TT,1,XL-3);
    IF (DEBUG=1) THEN DO; PUT SKIP;
 PUT SKIP LIST('XL=',XL,'TT NEAR LLL',TT);
  END;ELSE;
 LLL: LABEL2=L300;
              STATV(VERT)=2;
 END EXTEND;


 BACKUP:PROCEDURE;
 IF(LENGTH(TT)=0) THEN DO;LABEL3=L100;RETURN;END;
    ELSE LABEL3=L200;

          TT=SUBSTR(TT,1,(LENGTH(TT)-3));
 PUT SKIP LIST(' FROM BACKUP TT=',TT);
 END BACKUP;


 ADDCYCL:PROCEDURE;
          DECLARE (S1,S2,S3,S4,S5)CHARACTER(3*(NSTR+NODES))VARYING,
                  (XN,XNL) FIXED DECIMAL (2,0) ;
          S1='N'||ENCODE(VERT);
          XN=INDEX(TT,S1);
          XNL=LENGTH(TT);
          S2=SUBSTR(TT,XN+3,XNL-XN-2);
          S3=S1||S2;
          S4=S3||S1;
          CYC(CYCN)=S4;
    IF (DEBUG=1) THEN DO; PUT SKIP;
 PUT SKIP LIST(' FROM ADDCYCL ---TT=',TT); PUT SKIP LIST('S1',S1);
 PUT SKIP LIST('XN=',XN);PUT SKIP LIST('XNL',XNL);PUT SKIP LIST('S2',S2
);PUT SKIP LIST('S3',S3);PUT SKIP LIST('S4',S4);PUT SKIP LIST('CYC=',
CYC(CYCN));  PUT SKIP LIST(CYCN);
  END;ELSE;
          CYCN=CYCN+1;
 END ADDCYCL;

 EXAMINE:PROCEDURE;
    IF (DEBUG=1) THEN DO; PUT SKIP;
 PUT SKIP LIST('FROM EXAMINE--STATV=',STATV);
 PUT LIST('  STRM#  SV    TV  STATUS')SKIP(2);DO K1=1 TO NSTRMS;DO K2=1
```

```
TO 4;PUT EDIT(XREF(K1,K2))(X(3),F(2),X(4),F(2),X(3),F(2),X(4),F(2));
END;END;
 END;ELSE;

L1:     DO I=1 TO NSTR;
        IF(XREF(I,1)=LASTARC) THEN
   L2:DO;
        IF(STATV(XREF(I,3))=0) THEN
L3:DO;
          TT=TT||('N'||ENCODE(XREF(I,3)));
          STATV(XREF(I,3))=1;
        VERT=XREF(I,3);
          LABEL4=L2;
  IF (DEBUG=1) THEN DO; PUT SKIP;
PUT SKIP LIST('TT=',TT); PUT SKIP LIST('VERT=',VERT);
END;ELSE;
          RETURN;
      END L3;
        IF (STATV(XREF(I,3))=1) THEN
      L4:DO;
             LABEL4=L4;
       VERT=XREF(I,3);
  IF (DEBUG=1) THEN DO; PUT SKIP;
PUT SKIP LIST(' LOOP4-L4 VERT',VERT);
END;ELSE;
             RETURN;
        END L4;
 ELSE IF(STATV(XREF(I,3))=2) THEN
     L5:DO;
          LABEL4=L5;
       VERT=XREF(I,3);
  IF (DEBUG=1) THEN DO; PUT SKIP;
PUT SKIP LIST('LOOP L5 VERT=',VERT);
END;ELSE;
          RETURN;
     END L5; ELSE;
END L2;
 ELSE;
 END L1;
END EXAMINE;

CONCAT:PROCEDURE(STRG) RECURSIVE;

DECLARE  (CYTN ) FIXED DECIMAL(2,0);
 CYTN=1  ;
        DECLARE STRG CHARACTER(*) VARYING;
        DECLARE (CC,CC1,IPT,K,KK,LT)FIXED DECIMAL(2,0);
```

```
        DECLARE V CHARACTER(2),                    (C13,TEST)CHARACTER(3),
        (TAIL,C12,C11,C14,CY)CHARACTER(3*(NODES+NSTR))VARYING;
        DECLARE(LTT,LTT1,LTT2,LTT3,LTT4,LTT5,LTT6 ) FIXED DECIMAL(3,0);
 DECLARE  TEST1 CHARACTER(3);
 DECLARE ENDCP CHARACTER(3);
 DECLARE XJX CHARACTER(2);
 DECLARE (CYTAIL(50))CHARACTER(3*(NODES+NSTR))VARYING CONTROLLED;
  ALLOCATE CYTAIL  ; CYTAIL(*)='' ;

   IF (DEBUG=1) THEN DO; PUT SKIP;
 PUT SKIP LIST(' *****ENTERING CONCAT WITH RECUR=',RECUR); PUT LIST
(STATV);PUT SKIP LIST(CYC); PUT SKIP LIST(CYTAIL);
 PUT LIST('  STRNM  SV   TV  STATUS')SKIP(2);DO K1=1 TO NSTRMS;DO K2=1
 TO 4;PUT EDIT(XREF(K1,K2))(X(3),F(2),X(4),F(2),X(3),F(2),X(4),F(2));
 END;END;
  END;ELSE;

        LS=LENGTH(STRG);
        TEST1=SUBSTR(STRG,LS-2,3);
        V=SUBSTR(TEST1,2,2);
        CC1=CYCN-1;
 L1:    DO CC=1 TO CC1;
        IPT=INDEX(CYC(CC),TEST1);
        IF((IPT=0)|(IPT=1)) THEN GO TO ENDL1;
  ELSE;
     LTT=LENGTH(CYC(CC))-IPT-2;  /* ALSO=LENGTH OF TAIL*/
        TAIL=SUBSTR(CYC(CC),(IPT+3),LTT) ;
 IF (CYTN=1) THEN DO;CYTAIL(1)=TAIL; CYTN=2;GO TO CHK;END;
ELSE;
 L4:    DO K=1 TO (CYTN-1);
 IF(CYTAIL(K)=TAIL) THEN GO TO ENDL1;        ELSE;
  END L4;
        CYTAIL(CYTN)=TAIL;CYTN=CYTN+1;
 /* CHECK WHETHER TAIL HAS ANY VERTICES ON STRG*/
 CHK:
        LT3=LTT/3;
            L51:DO KK=1 TO LT3 BY 2;
                TEST=SUBSTR(TAIL,(3*KK+1),3);
                IF(INDEX(STRG,TEST)^=0)THEN GO TO ENDL1;
             ELSE; END  L51;

      ENDCP=SUBSTR(CYC(CC),LENGTH(CYC(CC))-2,3);
  XJX=SUBSTR(ENDCP,2,2);
 IF(STATV(DECODE(XJX))=2) THEN
 L6:DO; RECUR=1;
   CALL CONCAT(STRG||TAIL);
        GO TO ENDL1;  END L6;
```

```
  ELSE;
        LTT1=LENGTH(CYC(CC));
        C11=SUBSTR(CYC(CC),LTT1-2,3);
        LTT2=INDEX(TT,C11)+3;
        LTT3=LENGTH(TT);
        LTT4=LTT3-LTT2;
 C12=SUBSTR(TT,LTT2,LTT4+1);
        C13=TEST1;
        LTT5=INDEX((CYC(CC),C13) + 3 ;
        LTT6=LTT1 - LTT5;
        C14=SUBSTR(CYC(CC),LTT5,LTT6+1);
        CY=C11||C12||STRG||C14 ;

   IF(RECUR=,) THEN DO; CYC(CYCN)=CY;CYCN=CYCN+1;
PUT SKIP LIST(' FROM CONCAT--CYCLE=,CYCN=' ,CYC(CYCN-1),CYCN);
  END;
   ELSE;
        DO KKX=1 TO (CYCN-1) BY 1;
        IF(CY=CYC(KKX)) THEN GO TO ENDL1;
        ELSE;END;
PUT SKIP LIST(' CYCN FRM 294=', CYCN);
        CYC(CYCN)=CY; CYCN=CYCN+1;GO TO ENDL1;

     ENDL1: END L1;
 FREE CYTAIL;
  END CONCAT;
LSTOP: PUT PAGE;PUT SKIP LIST('  FINAL ANSWERS'); DO K=1 TO CYCN-1;
PUT SKIP LIST(CYC(K)); END;
 FREE  TT;  ALLOCATE LOOPCY;
        LOOPCY(*,*)=0;
        DO I=1 TO CYCN;L=LENGTH(CYC(I)); DO J=5 TO L BY 6;
T2=SUBSTR(CYC(I),J,2); XX=DECODE(T2); LOOPCY(I,XX)=1; END;
DO II=1 TO NSTRMS;PUT EDIT(LOOPCY(I,II))(80(X(1),F(1)));
END; PUT SKIP(2);
END;
/*THE NODE LOOPS ON THE SIGNAL FLOW GRAPH ARE THE STREAM LOOPS
IN THE ORIGINAL PROBLEM                                    */
/* THIS FOLLOWING SECTION SPECIFIC FOR THIS PBM ONLY */
ALLOCATE TEMPO; TEMPO(*)=0;
PUT SKIP LIST('  CONVR. TO ORIGINAL STRM NOTATIONS') ;
PUT SKIP(5);
DO I=1 TO CYCN; L=LENGTH(CYC(I));K=0;DO J=2 TO L BY 6;K=K+1;
T2=SUBSTR(CYC(I),J,2);XX=DECODE(T2);TEMPO(K)=XX;END;DO LX=1 TO K;
PUT EDIT(TEMPO(LX))(X(2),30(X(2),F(2)));END;PUT SKIP(2);END;
/* END OF SECTION */
        END M1;
ENCODE:PROCEDURE(LL) RETURNS(CHARACTER(2));
```

```
        DECLARE CC CHARACTER(2);
        DECLARE(LL,DIGIT1,DIGIT2) FIXED DECIMAL(2,0),
  CHR(0:9)CHARACTER(1) INITIAL('0','A','B','C','D','E','F','G','H',
 'I'   );
        DIGIT1=TRUNC(LL/10);
        DIGIT2=LL-DIGIT1 * 10;
        CC=CHR(DIGIT1)||CHR(DIGIT2);
        RETURN(CC);
END ENCODE;
DECODE:PROCEDURE(CC) RETURNS(FIXED DECIMAL(2,0));
        DECLARE (FCH,SCH) CHARACTER(1),CC CHARACTER(2),
  CHR(0:9)CHARACTER(1) INITIAL('0','A','B','C','D','E','F','G','H',
 'I'   );
        DECLARE (I,LX) FIXED DECIMAL(2,0);
        FCH=SUBSTR(CC,1,1);
        SCH=SUBSTR(CC,2,1);

L1:     DO I=0 TO 9 BY 1;
        IF(FCH=CHR(I)) THEN DIGIT1=I;ELSE;
        IF(SCH=CHR(I)) THEN DIGIT2=I;
   END L1;
        LX=DIGIT1*10 + DIGIT2;
    RETURN(LX);
END DECODE;
  END CYCLES;
```

1	2	1								
1	3	4	2	1						
3	4	3								
7	8	7								
1	3	4	7	8	2	1				
3	4	7	8	3						
4	7	8	4							
4	7	8	5	6	4					
5	6	5								
7	8	5	6	10	8	7				
1	3	4	7	9	5	6	10	8	2	1
3	4	7	9	5	6	10	8	3		
9	5	6	10	9						
7	8	10	8	7						
1	3	4	7	9	10	8	2	1		
3	4	7	9	10	8	3				
9	10	9								
1	3	10	8	7	9	4	2	1		
3	10	8	7	9	4	3				
1	3	10	8	7	9	5	6	4	2	1
3	10	8	7	9	5	6	4	3		
1	3	10	8	2	1					
3	10	8	2							
1	3	10	9	4	2	1				

Stream loops in Figure 8.2

3	1?	9	4	3						
1	3	1?	9	4	7	8	2	1		
3	1?	9	4	7	8	3				
1	3	1?	9	5	6	4	2	1		
3	1?	9	5	6	4	3				
1	3	1?	9	5	6	4	7	8	2	1
3	1?	9	5	6	4	7	8	3		
1	3	5	6	4	2	1				
3	5	6	4	3						
1	3	5	6	4	7	8	2	1		
3	5	6	4	7	8	3				
1	3	5	6	4	7	9	10	8	2	1
3	5	6	4	7	9	10	8	3		
1	3	5	6	1?	8	7	9	4	2	1
3	5	6	1?	8	7	9	4	3		
1	3	5	6	1?	8	2	1			
3	5	6	1?	8	3					
1	3	5	6	1?	9	4	2	1		
3	5	6	1?	9	4	3				
1	3	5	6	1?	9	4	7	8	2	1
3	5	6	1?	9	4	7	8	3		
1	7	8	2	1						
1	7	8	3	4	2	1				
1	7	8	3	1?	9	4	2	1		
1	7	8	3	1?	9	5	6	4	2	1
1	7	8	3	5	6	4	2	1		
1	7	8	3	5	6	1?	9	4	2	1
1	7	9	4	2	1					
1	7	9	4	3	10	8	2	1		
1	7	9	4	3	5	6	10	8	2	1

Stream loops in Figure 8.2 contd.

1	7	9	5	6	4	2	1			
1	7	9	5	6	4	3	10	8	2	1
1	7	9	5	6	10	8	2	1		
1	7	9	5	6	10	8	3	4	2	1
1	7	9	10	8	2	1				
1	7	9	10	8	3	4	2	1		
1	7	9	10	8	3	5	6	4	2	1

Stream loops in Figure 8.2 contd.