AN ABSTRACT OF THE THESIS OF

GEORGE D. STAVRAKIS for the DOCTOR OF PHILOSOPHY
(Name) (Degree)

in INDUSTRIAL ENGINEERING presented on August 21, 1980
(Major Department) (Date)

TITLE: A GENERAL LINEAR OPTIMIZATION ALGORITHM BASED UPON LABELING AND FACTORIZING OF BASIC PATHS ON RPM NETWORK

Redacted for privacy

Approved: _____
Dr. Michael S. Inoue

A new method is developed for solving linear optimization problems based on the RPM network modeling technique which represents the primal and the corresponding dual models simultaneously upon a single graph. The network structure is used to eliminate the need for explicit logical variables and to provide a graphic tool in analyzing the problem.

The new algorithm iterates through a finite number of basic solutions working towards optimality (primal) or towards feasibility (dual). At each iteration a set of critical constraints and basic structural variables are identified to form the current basic path network. A solution for the basic variables is obtained through factorization of the basis and used to update the nonbasic network. If the Kuhn-Tucker conditions are not satisfied, the method proceeds with the next iteration unless an unbounded or infeasible solution is encountered.

Under the new scheme, the original data remains unchanged throughout the optimization procedure and round-off errors can be kept to a minimum. Furthermore, the basic paths representation used in factorization reduces computer core requirement and permits direct-addressing of pertinent non-basic node data on disk storage. These features are especially appealing in solving large-scale problems even on limited computer hardware.

Since the size of the basis is never greater than the size of the basis required by simplex-type algorithms, the new scheme has an advantageous memory storage requirement.

Any basic solution (not necessarily optimum or feasible) can be used as a starting point and multipivoting can accelerate the optimization process.

In general, the number of iterations and the amount of operations depends on the sparsity of the constrained matrix and the complexity of the problem.

Statistical data based on sample experimental results indicate that the new algorithm, on the average, requires less arithmetic operations and no more iterations to reach the final solution than the simplex-type algorithms.

A GENERAL LINEAR OPTIMIZATION ALGORITHM BASED UPON LABELING AND FACTORIZING OF BASIC PATHS ON RPM NETWORK

by

GEORGE D. STAVRAKIS

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

Completed August 1980

Commencement June 1981

APPROVED:

Redacted for privacy

Professor of Industrial and General Engineering
in charge of major

Redacted for privacy

Head of Department of Industrial and General Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented August 21, 1980

ACKNOWLEDGEMENTS

I would like to take this opportunity, to express my graditute to those who made this thesis possible.

Many thanks to Dr. Michael S. Inoue. I feel very fortune to have him as my advisor. During the last two years he spent a lot of his time for this research. To Dr. James L. Riggs I want to express my appreciation for this invaluable encouragement and guidance throughout this research.

To Drs. David A. Butler and James W. Funck I feel very indebted for their recommendations and constructive criticisms. I would also like to thank Drs. Ronald B. Guenther, Harvey A. Meyer, Joel Davis and Robert D. Stalley for their assistance.

I would also like to thank Miss Cindy Tait who typed this thesis under a very tight schedule. I want to express my appreciation to the staff of Productive Resources Inc. for their cooperation and for allowing me to use their facilities.

TABLE OF CONTENTS

Chapter		Page
I	INTRODUCTION	1
	The Linear Programming Problem	1
	The Need for New Algorithms	1
	1. Memory space requirements	3
	2. Number of iterations	3
	3. Arithmetic operations per iteration	3
	4. Round-off errors	3
	5. Limited computer requirements	4
	6. Network structure	4
	The New Methodology	4
	1. Labeling	4
	2. Factorization	5
	3. Balancing	5
	Outlines of the Thesis	7
II	THE STATE OF THE ART	9
	Introduction	9
	Simplex Methods	9
	Dual Simplex Methods	12
	Primal-Dual Methods	12
	Generalized Inverse	14
	Matrix-Gactorization Methods	15
	1. The LDL^T factorization	15
	2. The LQ factorization	16
	3. The LU factorization	16
	Khachian's Algorithm to Linear Programming	16
	Surrogated Linear Programming	19
	Summary	21
III	RPM SET ANALYSIS	22
	Introduction	22
	Resource and Process Graph	22
	Resource and Process Nodes	25
IV	LABELING	30
	Introduction	30
	Types of Labeling	30
	1. Type (a)	30
	2. Type (b)	31
	3. Type (c)	31
	Type (a) Labeling	35
	Example 4.1	37
	Type (b) Labeling	40
	Example 4.2	43
	Type (c) Labeling	45
	Example 4.3	48
	Termination Conditions	50
	Interpretation of Basis in RPM	51

V	FACTORIZATION	52
	Theoretical Background	53
	The Decomposition Method	57
	The Labeled Process Graph Decomposition	57
	Example 5.1 (Process entry)	60
	The Labeled Resource Graph Decomposition	63
	Example 5.2 (Resource entry)	66
	Partial Factorization	69
	Type (1) Factorization	69
	Type (2) Factorization	72
	1. Process exchange	72
	2. Resource exchange	73
	Type (3) Factorization	73
	Discussion	74
VI	BALANCING	76
	Introduction	76
	The Process of Balancing	76
	Illustrative Example 6.1	79
	Illustrative Example 6.2	88
VII	COMPUTATIONAL RESULTS	99
	The Worst Case Analysis	99
	The Number of Operations per Iteration	101
	1. Labeling	103
	2. Factoring	104
	3. Balancing	104
	Statistical Data	104
VIII	CONCLUSION	108
	Feasibility of the New Algorithm	108
	Evaluation of the Criteria	109
	1. Memory space requirements	109
	2. Number of iterations	110
	3. Arithmetic operations per iteration	110
	4. Round-off errors	111
	5. Limited computer requirements	112
	6. Network structure	112
	Accelerating Methods	112
	Recommendations for Future Research	114
	1. The implementation of the factorization	114
	2. Analysis of network paths	114
	3. Approximation method	114
	4. Cycling	115
	5. Computational efficiency.	115

BIBLIOGRAPHY

APPENDIX A
RPM Notation
APPENDIX B
Listings of Computer Programs
APPENDIX C
Data Files
APPENDIX D
Sample Runs

LIST OF FIGURES

Figure		Page
1-1	The Basic Steps of the Algorithm	6
3-1	The RPM Network Representation	23
3-2	The RPM Network for Example 3.1	25
4-1	The Basic Steps at Each Iteration	33
4-2	An Example for Type (a) Labeling	39
4-3	An Example for Type (b) Labeling	44
4-4	An Example for Type (c) Labeling	49
5-1	The Basis Factorization Scheme	56
5-2	The Labeled Process Graph Decomposition	61
5-3	The Labeled Resource Graph Decomposition	67
5-4	Adding a New Pair of Nodes	71
6-1	The Initial RPM Network for Example 6.1	80
6-2	The First Iteration for Example 6.1	82
6-3	The Optimum Network for Example 6.1	85
6-4	Different Tableau for Example 6.1	87
6-5	The Initial RPM Network for Example 6.2	89
6-6	The First Iteration for Example 6.2	91
6-7	The Second Iteration for Example 6.2	93
6-8	The Final RPM Network for Example 6.2	95
6-9	Different Tableau for Example 6.2	97
7-1	All Possible Basic Solutions for an m x m Model	102

LIST OF TABLES

Table		Page
7-1	Executing Times for WANG 2200 B Model	100
7-2	Number of Iterations Required for the Sample Problems	106
7-3	Number of Operations Required for the Sample Problems	107

A GENERAL LINEAR OPTIMIZATION ALGORITHM BASED UPON LABELING AND FACTORIZING OF BASIC PATHS ON RPM NETWORK

I. INTRODUCTION

The Linear Programming Problem

The general linear programming problem can be described as follows: Given a set of m linear inequalities or equations in n variables, we wish to find non-negative values for these variables which will satisfy the constraints and maximize or minimize some linear function of the variables.

Mathematically, the problem is stated as:

$$\text{Maximize } z = \sum_{j=1}^{n} c_j x_j, \text{ such that } \sum_{j=1}^{n} a_{ij} x_j \ (\leq, =, \geq) \ b_i$$

$$x_j \geq 0; \ i = 1, 2, \ldots, m; \ j = 1, 2, \ldots, n$$

The widely documented simplex method, developed by George Dantzig (1963), was the first method presented for solving general linear programming problems.

The Need for New Algorithms

There have been several algorithms, most of them closely related to the simplex method, that attempt to reduce the computational effort involved in solving linear programming problems. The computational effort is widely classified into two categories; polynomial time and exponential time. An algorithm is said to run in polynomial time if the upper bound of the total amount of computations required is a polynomial function of the size of the problem. It is said to run in exponential time if the bound is an exponential function. In spite of several so called "polynomial time" algorithms,

including the Khachian's method discussed in Chapter Two, there has not been a practical algorithm that solves the general linear programming problem in polynomial time. Even more critical is the problem associated with the computer memory size requirement to implement currently available standard linear programming methods to large scale problems.

The simplex method is an exponential time algorithm that is not easily applicable to large size problems. In his article, "Linear Programming: Its Past and its Future," Dantzig (1977) wrote the following:

> Integrated national economy-energy models are currently being developed and solved using standard linear programming methods. Already a bottleneck on the size of energy models has been encountered because large-scale solution techniques are not available for practical application.

From an engineering point of view, a solution procedure that minimizes memory requirement and computational efforts for solving most practical problems is more desirable than even a theoretically polynomial time algorithm that is usually inefficient for solving large-scale practical problems.

The purpose of the research presented in this dissertation is to create a general purpose linear programming algorithm that is essentially suited for solving practical problems. Such an algorithm should first be able to solve any problem definable as a linear programming model. Second, it should minimize both the computer core memory space requirement and the amount of computational effort necessary to arrive at the solution. Ideally, the average memory

and time requirements for such a method should not be greater than those of the existing algorithms even for solving the worst case problems. The criteria considered were:

1. Memory space requirements. The new algorithm should require no more memory space than the simplex type algorithms. In most practical problems, the requirements should actually be less. A typical "practical" problem, for the sake of this dissertation is:

   (1) having a fair number of variables and constraints, say 1,000 x 1,000,

   (2) having a sparsely populated constraint matrix, and

   (3) having a constraint matrix that is more or less technologically ordered[1].

2. Number of iterations. The number of iterations required for the optimization procedure should be finite and not greater than that of the simplex-type algorithms.

3. Arithmetic operations per iteration. At each iteration, the average number of necessary arithmetic operations should be no greater than that of simplex-type algorithms.

4. Round-off errors. Many of the well known linear programming algorithms require alteration of model representation at each iteration. This often results in accumulation of round-off errors.

---

1. The term "technologically ordered" is borrowed from the Critical Path Method. In the LP context it implies that the nonzero elements form diagonal blocks in the basis matrix (Jacobs, pp. 356, 1977).

5. Limited Computer Requirement. The algorithm should be applicable to a mini-computer using disk storage system.

6. Network Structure. A network structure may eliminate the need for explicit logical variables as well as provide a graphic tool in analyzing the structure of the problem.

The New Methodology

In order to satisfy the above criteria, the new methodology is based on the RPM network modeling technique. The RPM (Resource Planning and Management) system, developed by Inoue & Riggs (1972), represents both the primal and dual model of the optimization problem simultaneously upon a single graph. On the network, each structural variable, x_j, is represented by a square node (decision process) and each constraint, i, by a circle node (resource node). The coefficients $a_{ij} \neq 0$, of the constraint matrix are shown as solid arrows. The cost coefficients c_j, and the constant right hand sides, b_i are shown as dotted arrows (Appendix A).

A process node is considered basic if the corresponding primal structural variable is basic in the traditional simplex sense. A resource node is considered basic if the corresponding structural variable in the dual model of the linear programming is basic. Thus, any basic node will have a zero residue. A basic path is a path on the RPM network that connects basic processes and basic resources.

Basic steps of the proposed algorithm (Figure 1.1)

1. Labeling. Unlike the simplex method, the dimension of the

basis for the new approach is not the same from one iteration to the next. The basis on the network is represented by a set of process nodes and the corresponding set of critical resource nodes[2]. The number of basic nodes (which should be the same for both the process nodes and the resource nodes) identifies the dimensionality of the basis for each iteration. The labeling technique is used to keep track of the current basic variables and critical constraints. At each iteration, the labeling process makes the proper change on the set of basic nodes. At each iteration, the complementary slackness theorem applies to all nodes and the algorithm stops as soon as the Kuhn-Tucker conditions are satisfied or an unbounded or infeasible solution is encountered.

2. Factorization. A decomposition scheme is used to form a triangular basis for the labeled network in such a way that both primal and dual solutions can be obtained by "back substitution." Depending on the change in labeling, each triangular basis can be modified in a way that complete refactorization is not necessary at each iteration.

3. Balancing. At each iteration, the values of the basic variables for the primal and dual problem are updated first. The non-basic nodes are updated next, each one independently. If the Kuhn-Tucker conditions are not satisfied the algorithm proceeds with

---

2. The terminology "basic" or "basis" in this dissertation refers to a set of basic structural variables in the primal and dual models of LP. For further discussion on this, refer to p. 51.

the next iteration.

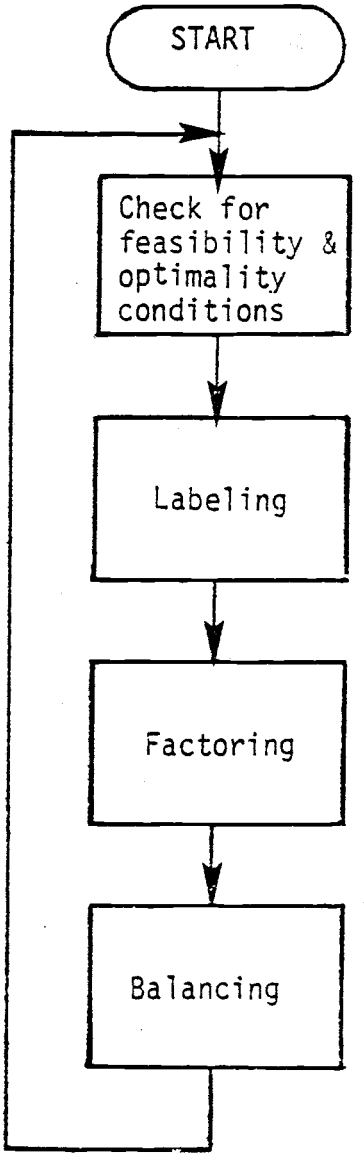


Figure 1-1. The Basic Steps of the Algorithm

Outlines of the Thesis

Chapter Two presents the general purpose linear programming algorithms. Attention will be directed to the "State-of-the-Art" algorithms, with emphasis on those which provide an exact solution to the problem. The polynomial time algorithm developed by L. G. Khachian is included. Among the algorithms that give an approximated solution to the problem, only recently developed algorithms will be discussed.

The discussion of the new methodology is included in Chapter Three through Eight.

Chapter Three introduces the RPM network structure and a Set Analysis for the primal and dual problem. The necessary network terminology is given in the same chapter.

The Labeling process is presented in Chapter Four. Three mutually exclusive and exhaustive types of labeling are discussed in detail. The need for different types of labeling is due to the fact that the algorithm operates on both the primal and dual model working toward optimality or toward feasibility. There is no restriction on the sequence in which the different types of labeling are applied. However, the number of iterations required for a given problem may depend upon the chosen sequence.

The decomposition algorithm used to triangularize the labeled network and the different factorization schemes corresponding to all possible changes during the labeling procedure are analyzed in Chapter Five.

Chapter Six gives the necessary computational procedure for balancing. The case in which all nodes of the network are updated at each iteration is considered here.

The theoretical computational effort for the algorithm and statistical data based on test problems are given in Chapter Seven. Chapter Eight includes the discussion on experimental results, conclusions, and recommendations for further studies.

The relationship between the traditional mathematical formulation of the linear programming problem and the RPM equivalent is included in Appendix A.

Appendix B contains listings of computer programs. Operating instructions for the software for mini-computers are given in Appendix C. Appendix D includes sample data and computer runs.

II. THE STATE OF THE ART

Introduction

The purpose of this chapter is to review the current status of the linear programming solution methods to establish the basis for discussing and evaluating the proposed methodology. Attention will be directed to the developments of the theory of linear optimization and not to the application. The different algorithms are divided into simplex-type algorithms and other methods. The simplex-type algorithms include simplex methods, dual methods, and primal-dual methods. Other methods include the generalized inverse method, matrix factorization methods, the Khachian's algorithm and the surrogated linear programming method. All of the above listed algorithms are "exact methods", with the exception of the surrogated linear programming method which is an "approximation method."

Only the more recent or especially significant contributions are noted. No attempt has been made to include all available techniques. J. W. Barnes and R. M. Crisp (1975) presented a detailed survey of general purpose LP algorithms.

Simplex Methods

The Simplex Method and the simplex algorithm for choosing the optimal feasible program was developed by the end of the summer of 1947. Intensive work began in June 1947 in an Air Force group that later was given the title of Project SCOOP (Scientific Computation

of Optimum Programs.)

The basic LP problem is:

$$\max Z = C^T X \text{ such that } AX \ (\leq, =, \geq)\ B,\ X \geq 0$$

where A is an mxn array, and C, B, X are column vectors.

The canonical form is given as:

$$\max Z = C^T X \text{ such that } AX \leq B,\ X \geq 0$$

and the dual problem:

$$\min Z = B^T Y \text{ such that } A^T Y \geq C,\ Y \geq 0,\ Y \text{ a column vector.}$$

Simplex Method and its best known refinement, the Revised Simplex, iterate from one extreme point of the feasible solution space to the next by changing the basic variables and increasing the objective function. Both techniques use the same criteria to determine the vector to enter the basis, the vector to leave the basis, and the terminal conditions. A comparison of the number of calculations needed to perform each step of an iteration points to the essential difference between the methods. If $n \geq 3m$ (where n is the number of columns, and m is the number of rows), the revised method requires less computation per iteration. Also, further savings can be made when the product form of the inverse is used. Other reasons that recommend the revised simplex method are:

1. It is simpler to introduce a new variable into the system after an optimum solution has been found.

2. The general solution to the dual problem seems more immediate.

3. Associated with the above reasons is the conclusion that

different post optimality problems and upper bounding techniques appear to be more easily solved (Wagner, 1956).

A technique developed by George Dantzig and Philip Wolfe (Dantzig and Wolfe, 1959) was the decomposition principle for linear programs. This technique permits the problem to be solved by alternate solutions of linear sub-programs representing its several parts and a coordinating program that is obtained from the parts by linear transformations. The coordinating program generates, at each cycle, new objective forms for each part, and each part generates in turn new activities for the interconnecting program. Besides its computational advantages, the principle of decomposition yields a certain rationale for the decentralized decision process in the theory of the firm.

Adi Ben-Israel and Philip D. Robers (1968) have developed a decomposition method for the interval linear programming.

The interval LP is of the form: max $C^T X$ such that $B^- \leq AX \leq B^+$. If A has full rank, the optimal solutions can be written explicitly. This result is used in conjunction with the decomposition principle to develop a finite iterative technique.

Generalized upper bounding techniques have been developed for handling constraints of the simple form $x_j \leq b_j$ implicitly without increasing the dimension of the problem. Problems with simple upper bounds often occur in practice.

A. Charnes and C. Lemke (1954) have introduced a modified simplex method. The modification consists of using, at each stage,

only the inverse matrix of the current basis and the original data. The main advantages of this method are: (1) the accumulation of round-off errors is confined entirely to the current inverse matrix and (2) there is no increase in computation due to the initially sparse matrix filling up with nonzero entries as one proceeds from stage to stage.

Dual Simplex Methods

The dual simplex algorithm simply allows the user to apply the simplex algorithm's rules (pivot selection) while maintaining the problem in its primal form.

C. E. Lemke (1954) developed a dual method for LP based on the exploitation of orthogonality and the dual theorem of Tucker, Kuhn and Gale. The resulting method yields the same advantages as the "modified simplex" method and also eliminates the need for doubling the number of variables where they are not necessarily nonnegative. This approach focuses on a simultaneous geometry of the primal and dual problems which may be advantageous in particular problems, and provides a visualization of the primal and dual at each stage, and also offers the possibility of a combination of the two methods.

Primal-Dual Methods

Primal-dual algorithms solve the primal and dual problems simultaneously. Perhaps the best known is the one by Dantzig,

Ford and Fulkerson. It is an extension to a primal-dual method in solving transportation problems. Artificial variables are added and a starting feasible solution to the dual must be available. At each iteration the dual feasibility will be preserved while iterating for primal feasibility. The algorithm stops as soon as a feasible solution has been found for the primal (Dantzig, Ford, and Fulkerson, 1956).

The algorithm of Talacko and Rockefeller (1960) and the MINIT algorithm of Llewellyn are similar. Given a problem with no primal or dual feasible solution, choose the primal or dual iteration satisfying the appropriate primal or dual iteration criteria that maximizes the absolute change in the objective function. All inequality constraints have implicit slack variables in the initial basis (all inequalities are less than or equal to, allowing negative right hand sides). The Gauss-Jordan method applied to the contracted tableau is used for changing the basis.

The criss-cross method developed by S. Zionts (1969) normally begins with a problem solution which is neither primal or dual feasible. The method generates a starting basic solution, then alternative primal and dual iterations are performed until primal and dual feasibility has occurred. The advantages of this method are (1) no feasible solution is required for the primal or dual, (2) no artificial variables are introduced, and (3) the product form can be employed.

M. Y. Harris (1970) has developed an algorithm which is similar

to that of Dantzig, Ford and Fulkerson in terms of using the complementary slackness concept and alternates between primal and dual problems. However, it is simpler, but requires a separate Phase I stage. The algorithm is initiated with basic feasible solutions to both the primal and dual problems: (max $Z = C^TX$ s.t. $AX + IS = B$), (min $u = W^TB$ s.t. $A^TW - IV = C$) which may be contained in the initial formulation of the problem or may be generated by Phase I of the simplex algorithm, then alternates between the primal and dual systems, optimizing the decision variables with respect to complementary slackness condition as the objective function.

Balinski and Gomory (1963) developed the Mutual Primal-Dual Algorithm. The algorithm employs a hierarchy of the subtableaux and uses a primal simplex pivot rule on subproblems until primal degeneracies occur and then applying a dual simplex pivot rule until the degeneracies are resolved.

Generalized Inverse

Let us consider a linear programming problem of the form:

maximize $z = CX$ subject to $AX \leq B$ where A is of full rank.

The main result is an explicit representation of the general solution in terms of a generalized inverse of A. The solution $A^{-1}B$ has possible computational advantages over simplex or other iterative methods of linear programming. Consider the following matrix equations:

$$ATA = A$$

$$TAT = T$$

$$(AT)^t = AT$$

$$(TA)^t = TA$$

where t means transpose.

The explicit solution is then of the form $X = TB + (I-TA)Y$, Y arbitrary and T such that $ATB = B$. The method is based on Moore-Penrose work on generalized inverse for A denoted by A^+ (Boullion, Odell, 1976).

Matrix-Factorization Methods

The way in which linear programming is performed commercially has been recently affected by established results in the field of Numerical Analysis. The way in which a factorization method is implemented depends upon the size of the problem it is designed to handle.

The three major factorizations used in linear algebra are:

1. The LDL^T factorization.

   Let $B = LDL^T$

   $B' = L'D'L'^T$; B is a square matrix where L, L' are lower-triangular matrices and D, D' are diagonal matrices. Usually, L and D are known and we want to find L' and D' when B changes to B'. A special case of practical importance is the so-called "rank-one modification" of B:

   $B' = B + \lambda YY^T$; where Y is a vector with norm 1, λ a real

number (Gill and Murray, 1977).

2. The LQ factorization.

Let $A = [L, 0]\ Q$

$A' = [L', 0]\ Q'$, where 0 is a zero array, and A, A' are n x m matrices (m > n), L is a lower-triangular and Q is such that $Q^T Q = 1$. Any array A can be factorized in this form. We want to find L' and Q' if we know L, Q and A changes to A' as:

$$A' = A + XY^T$$

3. The LU factorization.

Let $A = LU$

where L is a lower-triangular matrix (with unit diagonal elements) and U is an upper-triangular matrix. Let,

$$A' = A + XY^T$$

An explicit update of the LU factorization of A is required.

The advantage from a matrix factorization method is the computational savings when solutions to a sequence of related problems are required.

Khachian's Algorithm for Linear Programming

L. C. Khachian (1979) published a polynomial-bounded algorithm to check the solvability of a system of linear inequalities. It may be applied to solve linear programming problems in polynomial time.

The immediate importance of Khachian's results is theoretical. The new method may also be useful for a broader class of problems (nonlinear and dynamic programming problems). The new scheme, based on Shor's theoretical results (Shor, 1970) involves the construction of a sequence of ellipsoids in multidimensional space that close in on the optimal solution, when applied on linear programming problems. Many experts in the field, however, point out that the practicality of Khachian's method cannot be decided until much more computing experience with it has been obtained.

Khachian's method is tied to what is said to be the major unsolved problem in computer science, i.e. a polynomial-bounded algorithm for a class of problems known as NP-complete problems.

In 1970, N. Z. Shor presented a generalized method for minimizing a convex function f (x), defined over the entire n-dimensional Euclidean space. His method is a combination of the generalized gradient descent method and a transformation of the argument space (called space dilatation). (The usual gradient methods impose extra assumptions concerning the continuity of f (x)). Although he constructs the general algorithm, no attempt is made towards its practical behavior (round-off errors, etc.), or claim that the algorithm is polynomial bounded. Also, no application of his method to solve linear programming problems is explicitly stated. He explains, however, that the method can be applied in the more generalized case, i.e. to solve a system of equalities and convex inequalities.

For example let:

$$Fi\ (x) = 0;\ i = 1, 2, \ldots, r$$

$$Hj\ (x) < 0;\ j = 1, 2, \ldots, p$$

This can be reduced to a problem of minimizing:

$$f\ (x) = \max\ (0, \max\ Fi^{2}\ (x), \max\ Hj\ (x)\ )$$

Shor's algorithm is applicable to nonlinear programming (Shor, 1970).

The Algorithm.

Let $\sum_{j} a_{ij}\ x_j < b_i;\ i = 1, 2, \ldots, n.$ $j = 1, 2, \ldots, m.$

be a system of linear inequalities with integral coefficients.

Define:

$$L = \sum_{i,j} \log\ (|a_{ij}| + 1) + \sum_{i} \log\ (|b_i| + 1) + \log mn + 1$$

Let Aj; j = 0, 1, 2,... nxn matrices, and k; k = 0, 1, 2,.. n-dimensional column vectors.

Start with:

$x_0 = 0;\ A_0 = 2^{L}I$ (I is the identity matrix)

assuming that (x_k, A_k) is defined, compute t(x) such that:

$$t(x) = \max_{i}\ (a_i\ x - b_i);\ i = 1, 2, \ldots, m.$$

t(x) is the maximum discrepancy. If $t(x) < 0$ the current solution is optimal. Otherwise, along with t(x), i is defined from the above equation.

Next compute:

$$x_{k+1} = x_k - \frac{1}{n+1} \cdot \frac{A_k a_i}{a_i^T A_k a_i}$$

$$A_{k+1} = \frac{n^2}{n^2+1} \left(A_k - \frac{2}{n+1} \cdot \frac{(A_k a_i)(A_k a_i)^T}{a_i^T A_k a_i} \right)$$

If the system is inconsistent then the algorithm will stop after $16n^2L$ steps.

The L.P. case: If we want to solve the linear programming problem (in canonical form)

Maximize $c^T x$

s.t. $A x \leq b$ ($A = (a_{ij})$; $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$)

we consider the system of inequalities:

$$c^T x \leq b^T y$$

$$c^T x \geq b^T y \text{ (obj. fn)}$$

$$A x \leq b$$

$$x \geq 0 \text{ (primal)}$$

$$A^T y \geq c$$

$$y \geq 0 \text{ (dual)}$$

This is solvable if the original program has a feasible solution and a finite optimum. For any solution (x, y) of this system, x is an optimal solution of the L.P.

Surrogated Linear Programming

Surrogated linear programming is a different approach for

solving the general linear programming problem. The linear programming problem is replaced by an LP problem having a single constraint called the surrogate constraint. The surrogated constraint is a convex combination of the constraints of the original problem.

Glover used such constraints for the solution of zero-one integer programming problems (Glover, 1968). Staats also used the same type of constraints for solving the geometric programming problem (Staats, 1970). Greenberg and Pierskalla used surrogated constraints for the general L.P. problem (Greenberg and Pierskalla, 1969).

Any L.P. problem can be written in the form:

$$\max z = \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \leq P_i;\ i = 1,\ldots, m$$

where

$$P_i \equiv \begin{cases} 1 & i = 1,\ldots, k \\ -1 & i = k+1,\ldots, s \\ 0 & i = s+1,\ldots, m \end{cases} \qquad x_i \geq 0;\ j = 1,\ldots, n$$

The associated surrogated problem has the form:

$$\max z = \sum_{j=1}^{m} c_j x_j \quad \text{s.t.} \quad \sum_{i=1}^{m} \lambda_i \left(\sum_{j=1}^{n} a_{ij} x_j\right) \leq \sum_{i=1}^{k} \lambda_i - \sum_{i=k+1}^{s} \lambda_i$$

$$\sum_{i=1}^{m} \lambda_i = 0;\ 0 \leq \lambda_i \leq 1;\ x_j \geq 0;\ j = 1,\ldots, n$$

The characteristic of the surrogated programming is that the algorithm generates successive solutions which are interior points of the feasible region.

The algorithm itself (Dittmann, 1973) consists of a heuristic procedure for iterating from some starting vector of surrogate

multipliers (λ_i) to the optimum λ_i values. Since the surrogated problem has only one constraint there is a quick check for optimization for each set of λ_i values. The technique is not subject to round off errors and has promise for savings in computation time.

SUMMARY

Most of the "State of the Art" algorithms operate on a full basis (i.e. the dimensionality of the basis depends on the number of constraints or variables in the model).

Logical variables are added to the majority of the "exact solution" algorithms. The efficiency of the algorithm, for primal or dual simplex-type algorithms, is affected by the "shape of the problem," i.e. the ratio of the number of constraints and the number of variables.

All algorithms (with the exception of Khachian's algorithm) are exponential. This is the worst case where (m+n)!/m!n! is considered to be all possible solutions. Theoretically, Khachian's algorithm is polynomial time. However, experience so far has not been practical. All algorithms are based on matrix theory, and most of them are not distinguished between structural and logical variables. The original data is changed for the majority of the algorithms. Nonzero elements created in the basis is an important factor for the computational requirements and accuracy of the algorithm. Factorization methods alleviate the problem of nonzero elements created at each iteration.

III. RPM SET ANALYSIS

Introduction

The purpose of this chapter is to describe the RPM network terminology (Appendix A). First, the resource graph and process graph are defined. Each graph is a mapping of the nonzero elements of the constraint matrix on the RPM network, and remains unchanged throughout the optimization procedure. Next, we consider the set of nodes corresponding to the structural variables of the primal linear programming problem (process nodes), and the set of nodes corresponding to the constraints of the problem (resource nodes).

On the RPM network, each structural variable is represented by a square node. Given an RPM network with n process nodes (numbered one through n), we define J as the set of indexes of the process nodes: i.e. $J = \{1, 2, \ldots, n\}$. Also, given m resource nodes on the same network (numbered one through m), we define I as the set of indexes of the resource nodes: $I = \{1, 2, \ldots, m\}$

The ordered pair (k, u_k), where k is an element of J, $(k \in J)$, and u_k is a nonzero real number, is called "primal arc." The order pair (h, v_h), where h is an element of I, and v_h a nonzero real number, is called "dual arc."

Resource and Process Graph

Let us consider the LP model as defined in Chapter One.

Using the RPM notation (Appendix A), this problem may be restated as:

$$\max cx = \Sigma\, c_j^+ x_j - \Sigma\, c_j^- x_j \text{ subject to: } \Sigma a_{ij}^+ x_j - \Sigma a_{ij}^- x_j \leq b_i^+ - b_i^-$$

where, $x_j \geq 0$; $j = 1, 2, \ldots, n$, and $i = 1, 2, \ldots, m$. An example is given in Figure 3-1.

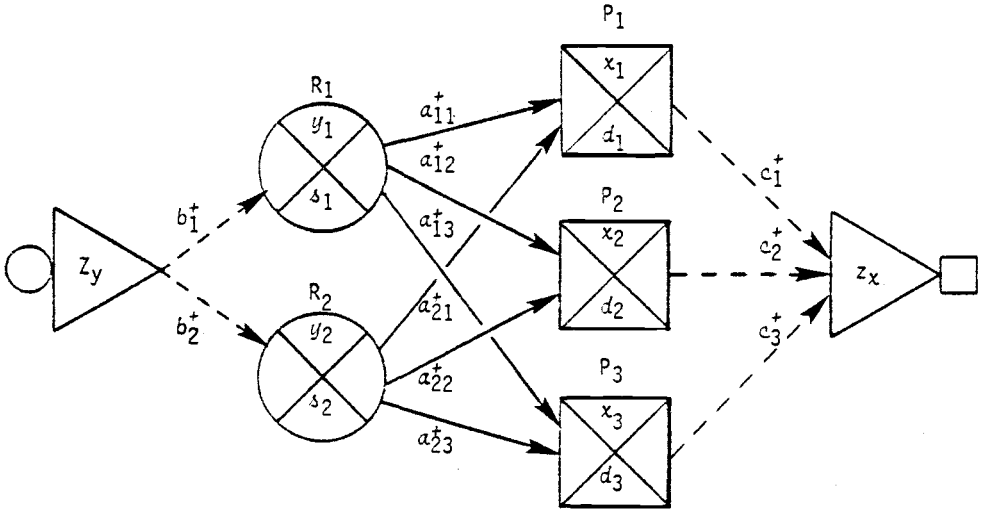


Figure 3-1. The RPM Network Representation.

For a given set of resource nodes I, and a set of process nodes J, we define:

The set I_i

The set I_i ;i = 1, 2,..., m, is the set of all process nodes which are connected to the resource node i. For i = 1, 2,..., m, let U_i be the set of all arcs corresponding to the resource node i and the set I_i, i.e.

$$U_i = \{(k, u_k);\ k \in I_i\};\ \text{where } u_k = a^+_{ik}$$

Def. 3.1. The set U defined as the union of all sets U_i i = 1, 2,..., m, is called "Resource Graph", i.e. $U = \cup U_i; i \in I$.

The set J_j

The set J_j, is defined as the set of all resource nodes which are connected to the process node j; j = 1, 2,..., n.

For j = 1, 2,..., n, let V_j be the set of all arcs corresponding to the process node j and the set J_j, i.e.

$$V_j = \{(h, v_h);\ h \in J_j\};\ v_h = a^+_{hj}$$

Def. 3.2. The set V defined as the union of all sets V_j j = 1, 2,..., n, is called "Process Graph," i.e. $V = \cup V_j;\ j \in J$. A primal (dual) arc is called "positive" if $u_k > 0$ ($v_h > 0$). Otherwise, it is called "negative," since zero elements are not allowed.

Let U_i^+ be the subset of U_i, which contains the positive arcs of U_i, a^+_{ij}, and U_i^- the subset of U_i which contains the negative arcs, a^-_{ij}, i = 1, 2,..., m. The sets V_j^+ and V_j^- are defined in the same way for j = 1, 2,..., n.

E.g. 3.1. For the RPM network shown in Figure 3-2 we have:

$$I = \{1, 2\};\ J = \{1, 2, 3\}$$

(i) Resource Graph:

$$U_1 = \{(1, 2), (2, 4)\};\ I_1 = \{1, 2\}$$

$$U_2 = \{(1, 3), (3, 5)\};\ I_2 = \{1, 3\}$$

and $U = \{U_1, U_2\}$ is the resource graph.

(ii) Process Graph:

$$V_1 = \{(1, 2), (2, 3)\};\ J_1 = \{1, 2\};$$

$$V_2 = \{(1, 4)\} \quad ;\ J_2 = \{1\};$$

$$V_3 = \{(2, 5)\} \quad ;\ J_3 = \{2\};$$

and $V = \{V_1, V_2, V_3\}$ is the process graph.

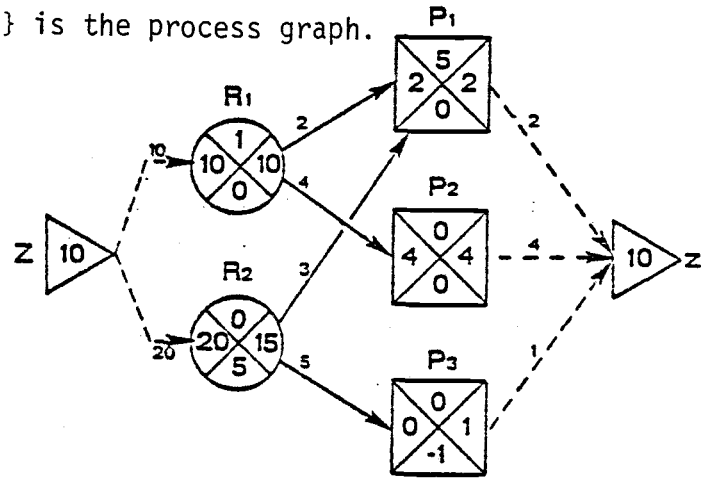


Figure 3-2. An RPM network for Example 3.1.

Resource and Process Nodes

Process Nodes

Let P_j denote the process node with index number j; $1 \leq j \leq n$.

Def. 3.3. For each process node P_j; $j = 1, 2, \ldots, n$.

(i) The variable X (j) is defined as the "process variable" corresponding to node j. The process variable is also called "primal variable" and corresponds to the structural

variable j of the linear programming problem.

(ii) The constant C (j) is defined as the "process constant," and corresponds to the cost coefficient, c_j^+, of the linear programming problem.

(iii) The constant D (j) is defined as "process residue" and corresponds to the $z_j - c_j^+$ coefficient of the linear programming simplex tableau.

Resource Nodes

Also, let R_i denote the resource node i; $1 \leq i \leq m$.

Def. 3.4. For each resource node R_i; i = 1, 2,..., m,

(i) The variable Y (i) is defined as the "resource variable" corresponding to the resource node i. The resource variable is also called "dual variable" and corresponds to the dual variable i, in the linear programming problem.

(ii) The constant B (i) is defined as the "resource constant," b_i^+, corresponding to the resource node i. The resource constant is the constant corresponding to the ith constraint of the linear programming problem.

(iii) The constant S (i) is defined as the "resource residue" of the resource node i, and replaces the slack or surplus variable of the corresponding linear programming problem.

Def. 3.5. For each resource node i, i = 1, 2,..., m, we define as "Resource Cluster i," the following set:

$$U_i' = \{R_i, U_i\}$$

Def. 3.6. For each process node j; j = 1, 2,..., n we define as "Process cluster j " the following set:

$$V_j' = \{P_j, V_j\}$$

Def. 3.7. For each resource node i; i = 1, 2,..., m, we define as "Resource Flow i" the following product:

$$(XU)_i = \Sigma u_k X(k);\ (k,u_k) \varepsilon U_i$$

Def. 3.8. For each process node j; j = 1, 2,..., n we define as "Process Flow j" the following product:

$$(YV)_j = \Sigma v_h Y(h);\ (h,v_h) \varepsilon V_j$$

Def. 3.9. Let $R = \{U, V, R, P, z, Z\}$ where,

U is the resource graph corresponding to the resource nodes

V is the process graph corresponding to the process nodes

R is the set of resource nodes

P is the set of process nodes

and z, Z are defined as

$$z = \sum_{j\varepsilon J} C(j)\, X(j);\quad Z = \sum_{i\varepsilon I} B(i)\, Y(i)$$

The set R we call an RPM network.

Def. 3.10. For any resource node, we define as "resource input" the following sum:

$$(XU)_i^+ = \sum_k u_k X(k) + B(i);\ (k, u_k) \varepsilon U_i^-;\ i = 1, 2,..., m$$

and as "resource output" the following sum:

$$(XU)_i^- = \sum_k u_k X(k);\ (k, u_k) \varepsilon U_i^+;\ i = 1, 2,..., m.$$

Def. 3.11. For any process node, the "process input" is defined as the sum:

$$(YV)_j^+ = \sum_h v_h Y(h);\ (h, v_h) \in V_j^+;\ j = 1, 2, \ldots, n$$

and as "process output" the following sum:

$$(YV)_j^- = \sum_h v_h Y(h) + C(j);\ (h, v_h) \in V_j^-;\ j = 1, 2, \ldots, n$$

Def. 3.12. As "resource residue" of any resource node, we define:

$$S(i) = (XV)_i^+ - (XV)_i^-;\ i = 1, 2, \ldots, m$$

Def. 3.13. As "process residue" for any process node is defined as:

$$D(j) = (YV)_j^+ - (YV)_j^-;\ j = 1, 2, \ldots, n$$

E.g. 3.2. For the RPM network shown in Figure 3.2, the resource and process nodes are:

(i) Resource nodes, R_i; $i = 1, 2$.

For $i = 1$, the resource variable $Y(1)$ is equal to 1, the resource constant $B(1)$ is equal to 10 and the residue $S(1)$ is equal to 0. The resource input is equal to $B(1)$, i.e. equal to 10, and the resource output is $u_1 \cdot X(1) = 2 \cdot 5 = 10$. For $i = 2$, we have:

$$Y(2) = 0,\ B(2) = 20 \text{ and } S(2) = 5$$

(ii) Process nodes, P_j; $j = 1, 2, 3$.

For $j = 1$, the process variable $X(1)$ is equal to 5, the

process constant C (1) is equal to 2 and the residue D (1) is 0. The process input is $v_1 Y(1) = 1 \cdot 2 = 2$, and the process output is C (1) = 2. For j = 2, we have: X (2) = 0, C (2) = 4 and D (2) = 0. Also, for j = 3: X (3) = 0, C (3) = 1 and D (3) = -1.

The primal objective function is equal to: $C(1) \cdot X(1) = 2 \cdot 5 = 10$, and the dual objective function Z is equal to $Z = B(1) \cdot Y(1) = 10 \cdot 1 = 10$.

IV. LABELING

Introduction

Labeling is the process that identifies the set of process and resource nodes to be included in the basis. The labeled sets of process and resource nodes, I^* and J^* grow from the initial nul sets by adding or sometimes deleting basic variables.

In this chapter, the labeling process will be described. The number of labeled resource nodes and the number of labeled process nodes will always be equal to each other. This number, ρ, is called the "dimension of the labeled network."

For simplicity we assume that only greater than or less than constraints are involved in the LP model in its canonical form. In this case, the labeling procedure will continue until the Kuhn Tucker conditions are satisfied, or in the absence of optimum solution, an indication of unboundedness or nonfeasibility will terminate the labeling.

At each iteration only certain changes in node labeling will occur. The following discussion applies in the case of single labeling. The case where multiple labeling accelerates the computational procedure, will be discussed in Chapter Eight.

Types of Labeling

1. Type (a). Any process node (unlabeled) with negative residue can be a candidate for labeling. For every newly labeled process node the Set Equality condition must be maintained. One of the two

cases are possible:

(a.1.) A labeled process node may be changed to an unlabeled node, thus maintaining the same size of dimensionality (ρ) for the labeled network, or

(a.2.) An unlabeled resource node may be changed to a labeled node, thus increasing ρ by one.

2. Type (b). Any unlabeled resource node with negative residue can be considered as a candidate for labeling. As in type (a) labeling, in order to maintain the Set Equality, one of the following must happen:

(b.1.) An unlabeled process is changed to labeled along with the candidate resource node (ρ increased by one), or

(b.2.) An already labeled resource node is unlabeled at the same time that the candidate becomes labeled (ρ remains the same).

3. Type (c). Any labeled resource or process node with a negative variable may be considered a candidate for unlabeling. In this case, the Set Equality requirements provides the following possible changes.

(c.1.) An already labeled process (resource) node is unlabeled (ρ is decreased by one), or

(c.2.) An unlabeled resource (process) node is labeled (ρ remains the same).

When the type (a) labeling is considered, we first look at the process cluster (the set consisting of the resource nodes connected to the candidate for labeling). We next perturbate the value of the candidate node variable. Increasing the value of the process variable by an arbitrarily small value (ε) allows us to observe the

changes of the variables that belong to the labeled network. We can increase the value of the candidate process variable as long as:

(a) The values of the nonnegative labeled process variables do not become negative, and

(b) Nonnegative resource residues, corresponding to unlabeled resource nodes, do not become negative.

From the above discussion it is clear that at least one variable (resource or process) will be driven to zero value, or an unbounded situation will arise.

Type (b) labeling requires the same procedure applied to the process graph.

For type (c) labeling we perturbate the value of the residue of the candidate node.

Any resource node that corresponds to an equal to constraint with zero residue is not a candidate for labeling. Otherwise, if the residue is negative, then the type (b) labeling is applied for that node. For a resource node corresponding to equal to constraint that has positive residue, the sign of the primal arcs and the constant b are changed. Again, type (b) labeling can be applied in this case. Type (c) labeling never applies to equal to constraints (negative value for variables corresponding to equal to constraints is allowed).

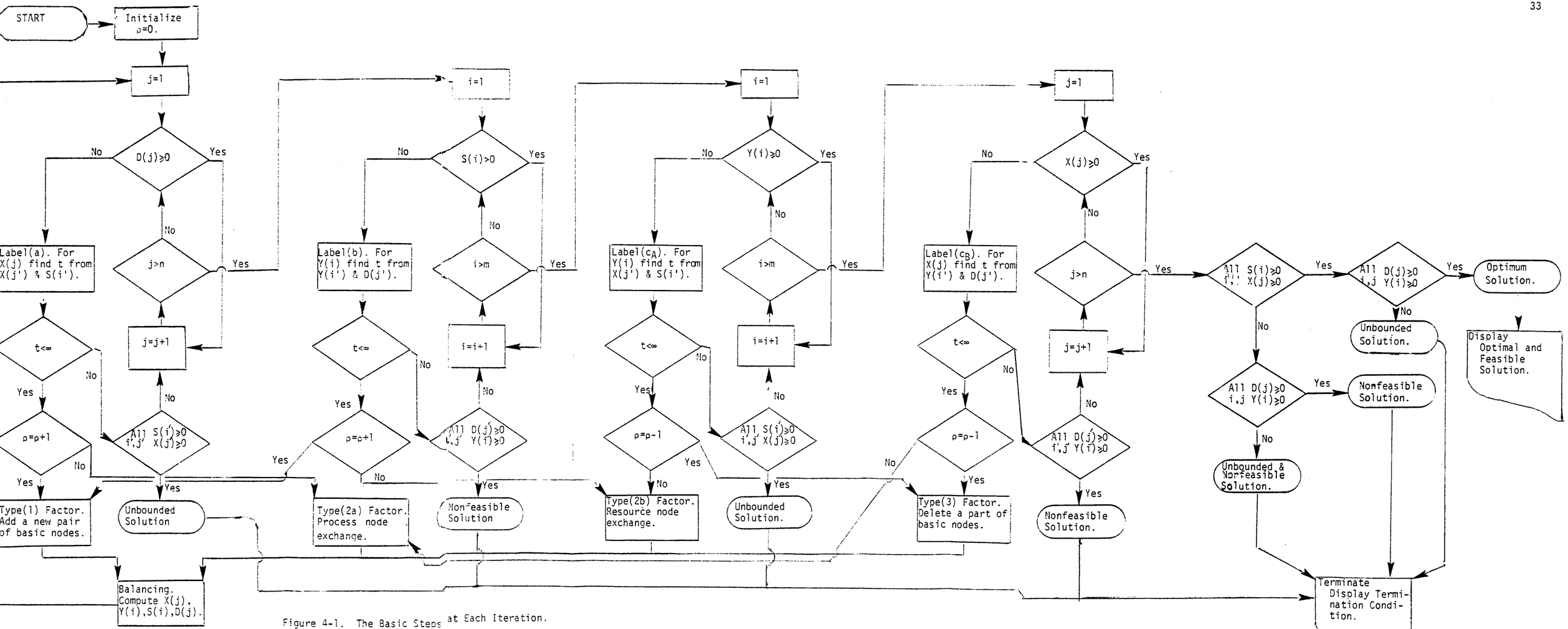


Figure 4-1. The Basic Steps at Each Iteration.

Notes to Figure 4-1

LP Simplex interpretation of RPM variables for an LP problem with n structural variables and m constraints:

X(j) = primal structural variable for the original LP problem.

D(j) = Simplex criterion $(z_j - c_j)$ corresponding to X(j), a logical variable.

Y(i) = Langrange Multiplier for the constraint i = the structural variable for the dual model of the original LP problem.

S(i) = Logical slack or surplus variable corresponding to the i^{th} constraint.

i = current index for the constraint i; $1 \leq i \leq m$ (pivot row).

j = current index for the primal variable j; $1 \leq j \leq n$ (pivot column).

i', j' = temporary indexes for i and j; $1 \leq i' \leq m$; $1 \leq j' \leq n$.

t = $\min(t_1, t_2)$.

t_1 = the θ ratio for pivoting in X(j) or Y(i) due to the structural variable ratio (e.g. $t_1 = \min(\varepsilon X(j)/|X_\varepsilon(j)|)$.

t_2 = the θ ratio for pivoting in X(j) or Y(i) due to the logical variable ratio (e.g. $t_2 = \min(\varepsilon S(i)/|S_\varepsilon(i)|)$.

ρ = the number of primal structural variables in the basis.

Type (a) Labeling

Consider a process node j_1 with negative residue, i.e. $D(j_1) < 0$. We define:

(i) the set J_{j_1} as the set of all resource nodes which are connected to the process node j_1. The set J_{j_1} is divided into:

(a) The subset $J^*_{j_1}$ which contains all the labeled resource nodes connected to the process node j_1, and:

(b) The subset $J^\circ_{j_1}$ which contains all the unlabeled resource nodes connected to the process node j_1, i.e. $J_{j_1} = J^*_{j_1} \cup J^\circ_{j_1}$.

(ii) the set V_{j_1} as the set of arcs which connect the process node j_1 and the resource nodes, i.e.

$$V_{j_1} = \{(h, v_h);\ h \in J_{j_1}\}$$

(iii) for each labeled resource node i ($i \in I^*$), the resource flow $(XU)_i$ as:

$$(XU)_i = \sum_k u_k X(k);\ k \in J^* \text{ and } (k, u_k) \in U_i$$

The residue of any labeled resource node is zero (complementary slackness theorem). Therefore, for a labeled resource node the resource flow is:

$$(XU)_i = B(i);\ i \in I^*$$

Let $X(j_1) = \varepsilon$ where ε is a positive number with an arbitrarily small value.

We can increase the value of $X(j_1)$ as long as:

(a) The values of the nonnegative labeled process variables do not become negative, and

(b) Non-negative resource residues, corresponding to unlabeled resource nodes, do not become negative.

Let t_1 be the maximum value allowed for $X(j_1)$ so that condition (a) is satisfied.

We define $X_\varepsilon(j);\ j \in J^*$ to be the change of the variable of the labeled process node j, when $X(j_1)$ is set equal to ε.

For each labeled resource node i, if $i \in J^*_{j_1}$

$$(X_\varepsilon U)_i = \sum_k u_k X_\varepsilon(k) = -\varepsilon u_{j_1};\ k \in J^* \qquad (4.1.)$$

otherwise:

$$(X_\varepsilon U)_i = \sum_k u_k X_\varepsilon(k) = 0;\ k \in J^* \qquad (4.2.)$$

Equations (4.1.) and (4.2.) give the values of the variables $X_\varepsilon(j);\ j \in J^*$.

Condition (a) is satisfied if:

$$X(j) + X_\varepsilon(j) \geq 0;\ j \in J^*$$

Therefore, the value of t_1 can be found as:

$$t_1 = \min_j \left\{ \varepsilon \frac{X(j)}{|X_\varepsilon(j)|};\ X_\varepsilon(j) < 0;\ X(j) \geq 0, \infty \right\} = \frac{X(j_2)}{X_\varepsilon(j_2)} \qquad (4.3.)$$

Let t_2 be the maximum value allowed for $X(j_1)$ so that condition (b) is satisfied. We define $S_\varepsilon(i)$ to be the change of the resource residue i, when the residue for that resource node is non-negative.

The changes of the non-negative resource residues are:

Example 4.1. Consider the network shown in Figure 4.2. The sets of the resource and process nodes are: $I = \{1, 2, 3\}$, $J = \{1, 2\}$. The labeled and unlabeled resource and process nodes are:

$$I^* = \{3\},\ I^\circ = \{1, 2\} \text{ and } J^* = \{1\},\ j^\circ = \{2\}.$$

The current solution $X(1) = 10$, $X(2) = 0$ is not optimum. The residue of the process node 2 is negative, $D(2) = -4 < 0$. Therefore, type (a) labeling is applied. The candidate for labeling is the process node $j_1 = 2$ for which:

$V_2 = \{(1,3), (2,1), (3,-1)\}$; $J_2 = \{1, 2, 3\}$; $J_2^* = \{3\}$; $J_2^\circ = \{1, 2\}$

(i) Let $X_\varepsilon(2) = \varepsilon$. Equation 4.1 becomes:

$$(X_\varepsilon U)_3 = 1 \cdot X_\varepsilon(1) = -(-1)\,\varepsilon, \text{ i.e. } X_\varepsilon(1) = \varepsilon$$

The value of t_1 defined from 4.3 is $t_1 = \infty$.

(ii) The updated positive resource residues are:

$$S_\varepsilon(1) = -3\varepsilon - 2\varepsilon = -5\varepsilon$$

$$S_\varepsilon(2) = -\varepsilon$$

The value of t_2 defined from Equation 4.4 is:

$$t_2 = \min_{1,2} \left\{ \frac{15}{5}, \frac{13}{1} \right\} = \frac{15}{5} = 3, \text{ i.e. } i_1 = 1$$

Finally, the value of t is:

$$t = \min(t_1, t_2) = \min(\infty, 3) = 3$$

Since $t = t_2$, resource node $i_1 = 1$, and process node $j_1 = 2$ are labeled.

$$S_\varepsilon(i) = -\sum_j u_j X_\varepsilon(j)$$

where $(j, u_j) \in U_i$; $j \in J^* \cup \{j_1\}$ and $i \in I^\circ$ such that $S(i) \geq 0$.

The value of t_2 is found from: $S(i) + S_\varepsilon(i) \geq 0$, or

$$t_2 = \min_i \{\varepsilon \frac{S(i)}{|S_\varepsilon(i)|};\ S(i) \geq 0;\ S_\varepsilon(i) < 0\} = \frac{S(i_1)}{S_\varepsilon(i_1)} \qquad (4.4.)$$

Otherwise, if $S_\varepsilon(i) \geq 0$ for all i, then $t_2 \equiv \infty$. Let $t = \min (t_1, t_2)$. Then:

(i) If $t = t_1 \leq t_2$ we label process node j_1 and unlabel process node j_2

(ii) If $t = t_2$ we label resource node i_1 and process node j_1. When process node j_2 is not uniquely defined from equation 4.3, then we arbitrarily choose any of the process nodes that gives the minimum ratio. Also, the resource node i_1 may not be uniquely defined from equation 4.4. In that case, we arbitrarily choose any resource node that gives the minimum ratio.

If t is ∞ then:

(i) If the current solution is infeasible then node j_1 cannot be a candidate for labeling.

(ii) If the current solution is feasible then the solution is unbounded.

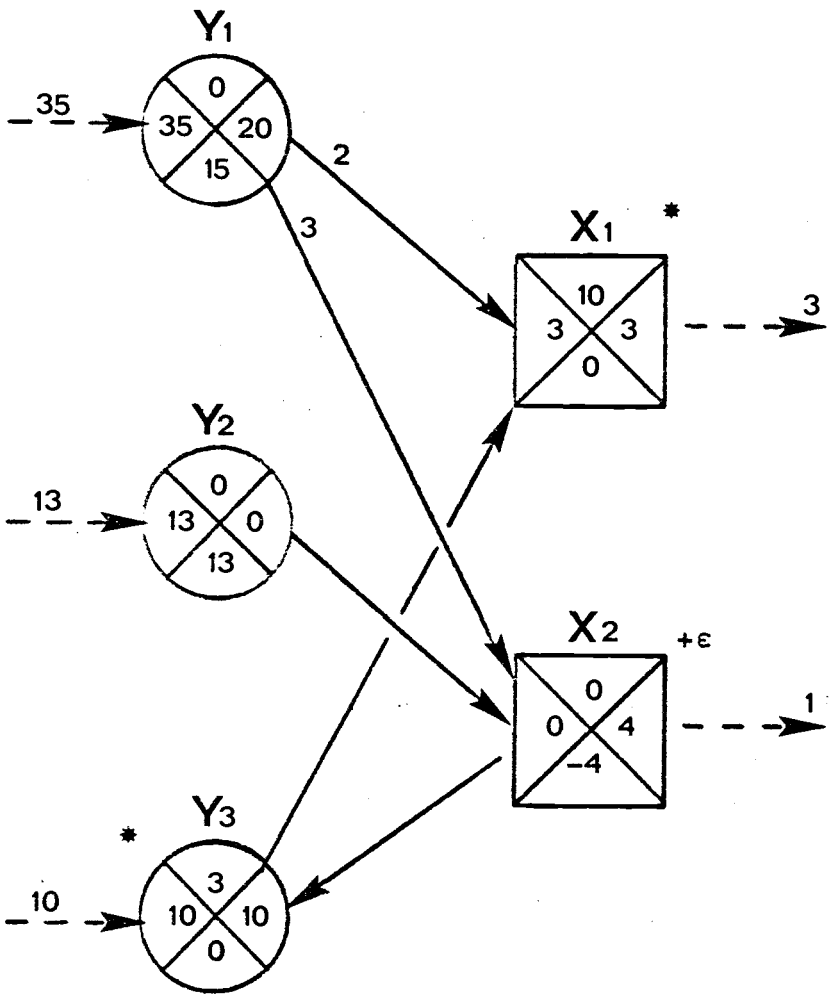


Figure 4-2. An Example for Type (a) Labeling.

Type (b) Labeling

Consider a resource node i_1 with negative residue i.e. $S(i_1) < 0$. We define:

(i) The set I_{i_1} as the set of all process nodes which are connected to the resource node i_1. The set I_{i_1} is divided into:

(a) The subset $I_{i_1}^*$ which contains all the labeled process nodes connected to the resource node i_1, and

(b) The subset $I_{i_1}^\circ$ which contains all the unlabeled process nodes connected to the resource node i_1.

i.e. $I_{i_1} = I_{i_1}^* \cup I_{i_1}^\circ$

(ii) The set U_{i_1}, as the set of arcs that connect the resource node i_1 and the process nodes, i.e.

$$U_{i_1} = \{(k, u_k);\ k \in I_{i_1}\}$$

(iii) For each labeled process node j ($j \in J^*$), the process flow $(YV)_j$ is defined as:

$$(YV)_j = \sum_h v_h Y(h);\ h \in I^*;\ (h, v_h) \in V_j$$

The residue of any labeled process node is zero i.e. $D(j) = 0;\ j \in J^*$. Therefore,

$$(YV)_j = C(j);\ j \in J^*$$

Let $Y(i_1) = \varepsilon$, where ε is a positive number arbitrarily small. We can increase the value of $Y(i_1)$ as long as:

(a) Any non-negative value of a labeled resource variable does not become negative and:

(b) Non-negative process residues corresponding to unlabeled process nodes do not become negative.

Let t_1 be the maximum value allowed for $Y(i_1)$ so that condition (a) is satisfied. Let $Y_\varepsilon(i)$; $i \in I^*$ be the change of the labeled resource variable $Y(i)$ when $Y(i_1)$ is set equal to ε.

For each labeled process node j ($j \in J^*$), if $j \in I_{i_1}^*$ then:

$$(Y_\varepsilon V)_j = \sum_h v_h Y_\varepsilon(h) = -\varepsilon v_{i_1} ; \; h \in I^* \qquad (4.5.)$$

Otherwise,

$$(Y_\varepsilon V)_j = \sum_h v_h Y_\varepsilon(h) = 0; \; h \in I^* \qquad (4.6.)$$

Equations 4.5 and 4.6 give the values of the variables $Y_\varepsilon(i)$; $i \in I^*$

The value of t_1 is found from: $Y(i) + Y_\varepsilon(i) \geq 0$, for all i such that $Y(i) \geq 0$, or

$$t_1 = \min_i \left\{ \frac{\varepsilon\, Y(i)}{|Y_\varepsilon(i)|} ; \; Y_\varepsilon(i) < 0 \right\} = \frac{Y(i_2)}{Y_\varepsilon(i_2)} \qquad (4.7.)$$

Let t_2 be the maximum value allowed for $Y(i_1)$ so that condition (b) is satisfied.

Then, we define $D_\varepsilon(j)$ to be the change of the process residue $D(j)$, for each process node j with non-negative residue. Then,

$D_\varepsilon(j) = \sum_i v_i Y_\varepsilon(i)$, where $(i, v_i) \in V_j$; $i \in I^* \cup \{i_1\}$ and $j \in J^\circ$ such that $D(j) \geq 0$. The value of t_2 is found from:

$D(i) + D_\varepsilon(i) \geq 0$, or

$$t_2 = \min \{\varepsilon \frac{D(j)}{|D_\varepsilon(j)|} ; D(j) \geq 0; D_\varepsilon(j) < 0\} = \frac{D(j_1)}{D_\varepsilon(j_1)} \quad (4.8.)$$

Otherwise, if $D_\varepsilon(j) \geq 0$ for all j, then $t_2 = \infty$. Let $t = \min (t_1, t_2)$. Then,

(i) If $t = t_1 \leq t_2$ we label resource node i_1 and unlabel resource node i_2.

(ii) If $t = t_2$ we label resource node i_1 and process node j_1.

When the resource node i_2 is not uniquely defined from equation 4.7, then we arbitrarily choose any resource node that gives the minimum ratio. Also, the process node j_1 may not be uniquely defined from equation 4.8. In that case, we arbitrarily choose any process node that gives the minimum ratio.

If t is ∞ then:

(i) If the optimality conditions are not satisfied then i_1 cannot be a candidate for labeling.

(ii) If the optimality conditions are satisfied then the dual problem is unbounded.

Example 4.2. Consider the RPM network shown in Figure 4-3.

The set of resource nodes I is, $I = \{1, 2, 3\}$ and the set of process nodes J, $J = \{1, 2\}$. The corresponding labeled sets I^* and J^* are:

$$I^* = \{2\}, \text{ and } J^* = \{1\}$$

The current solution is not feasible. The residue of the resource node 3 is negative, $S(3) = -5 < 0$, and type (b) labeling is applied:

The candidate for labeling is $i_1 = 3$, and for that resource node, $U_3 = \{(1,-1), (2,-2)\}$; $I_3 = \{1, 2\}$; $I_3^* = \{1\}$; $I_3^\circ = \{2\}$.

(i) Let $Y_\varepsilon(3) = \varepsilon$. Equation 4.5 becomes:

$(Y_\varepsilon V)_1 = -3Y_\varepsilon(2) = -\varepsilon \cdot (-1)$ or $-3Y_\varepsilon(2) = \varepsilon$ i.e. $Y_\varepsilon(2) = -\frac{1}{3}\varepsilon$.

The value of t_1 is found from equation 4.7 to be:

$$t_1 = \min_2 \{\varepsilon \frac{1}{|-1/3\varepsilon|} = 3, \text{ therefore } i_2 = 2.$$

(ii) The change of the non-negative process residues are:

$$D_\varepsilon(2) = -2 \cdot (-\tfrac{1}{3})\,\varepsilon - 2\varepsilon = -\tfrac{1}{3}\varepsilon.$$

The value of t_2 is found from 4.8 to be:

$$t_2 = \min_2 \{\varepsilon \frac{1}{|-4/3|\varepsilon}\} = \frac{3}{4}, \text{ therefore } j_1 = 2$$

The value of t is

$$t = \min\{t_1, t_2\} = \min\{3, \tfrac{3}{4}\} = \frac{3}{4}, \text{ i.e. } t = t_2$$

In this case we label resource node $i_1 = 3$ and process node $j_1 = 2$.

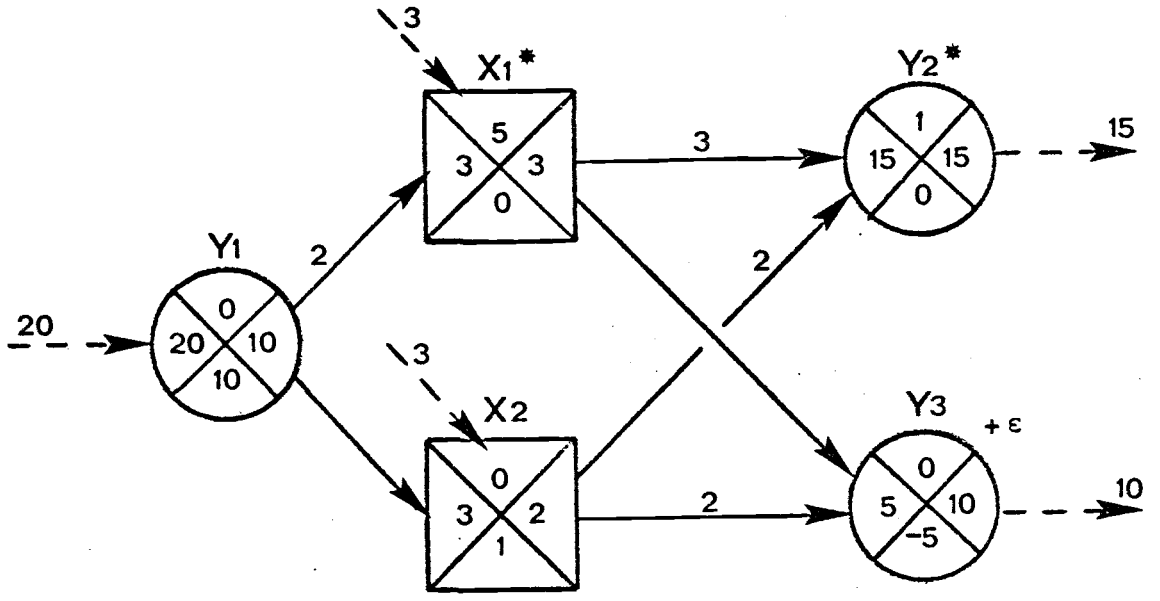


Figure 4-3. An Example for Type (b) Labeling.

Type (c) Labeling

A. Consider the case where there is a labeled resource node i_1, with $Y(i_1) < 0$. Let $S(i_1) = \varepsilon$ where ε is an arbitrarily small number.

In this case we increase the value of $S(i_1)$ as long as:

(a) The values of the labeled process variables do not become negative, and:

(b) Non-negative resource residues do not become negative.

Let t_1 be the maximum value allowed for $S(i_1)$ so that condition (a) is satisfied.

The perturbated values $X_\varepsilon(j)$; $j \in J^*$, for the labeled process variables are found from:

$$(X_\varepsilon U)_i = -\varepsilon \text{ for } i = i_1, \text{ and} \qquad (4.9.)$$

$$(X_\varepsilon U)_i = 0 \text{ for } i \in I^*;\ i \neq i_1 \qquad (4.10.)$$

In order to satisfy condition (a), i.e. $X(j) + X_\varepsilon(j) \geq 0$; $j \in J^*$, the value of t_1 is defined as:

$$t_1 = \min_j \{\varepsilon \frac{X(j)}{|X_\varepsilon(j)|};\ X_\varepsilon(j) < 0\} = \frac{X(j_1)}{X_\varepsilon(j_1)};\ X(j) \geq 0 \qquad (4.11.)$$

Let t_2 be the maximum value allowed for $S(i_1)$ so that condition (b) is satisfied.

The changes of the non-negative resource residues are:

$$S_\varepsilon(i) = \sum_j u_i X_\varepsilon(j) \text{ where}$$

$(j, u_j) \in U_i$; $j \in J^*$ and for all i such that $S(i) \geq 0$.

The value of t_2 is found as:

$$t_2 = \min \{\varepsilon \frac{S(i)}{|S_\varepsilon(i)|} ; S(i) \geq 0; S_\varepsilon(i) < 0\} = \frac{S(i_2)}{S_\varepsilon(i_2)}$$

Let $t = \min (t_1, t_2)$. (4.12.)

Then,

(i) If $t = t_1 \leq t_2$ we unlabel resource node i_1 and process node j_1.

(ii) If $t = t_2$ we label resource node i_2 and unlabel resource node i_1.

In the case where t_1 is not uniquely defined from equation 4.11, j_1 is chosen arbitrarily. In the same way, if t_2 is not uniquely defined from equation 4.12, i_2 is chosen arbitrarily. If $t = \infty$, then the same is true as in type (a) labeling.

B. Finally, consider the case where there is a labeled process node j_1, with $X(j_1) < 0$. Let $D(j_1) = \varepsilon$. In this case we increase the value of $D(j_1)$ as long as:

(a) The values of the labeled resource variables do not become negative, and

(b) Non-negative process residues do not become negative

Let t_1 be the maximum value allowed for $D(j_1)$ so that condition (a) is satisfied.

The perturbated values $Y_\varepsilon(i)$; $i \in I^*$, for the labeled resource variables are found from:

$$(Y_\varepsilon V)_j = +\varepsilon \text{ for } j = j_1 \text{ and} \quad (4.13.)$$

$$(Y_\varepsilon V)_j = 0 \text{ for } j \in J^*; j \neq j_1 \quad (4.14.)$$

In order to satisfy condition (a) i.e. $Y(i) + Y_\varepsilon(i) \geq 0$; $i \in I^*$ the value of t_1 is defined as:

$$t_1 = \min_i \{\varepsilon \frac{Y(i)}{|Y_\varepsilon(i)|} ; Y_\varepsilon(i) < 0\} = \frac{Y(i_1)}{Y_\varepsilon(i_1)}; Y(i) \geq 0 \qquad (4.15.)$$

Let t_2 be the maximum value allowed for $D(j_1)$ so that condition (b) is satisfied.

The changes of the non-negative process residues are:

$$D_\varepsilon(j) = \sum_i v_i Y_\varepsilon(i) \text{ where}$$

$(i, v_i) \in V_j$; $i \in I^{*i}$ and for all j such that $D(j) \geq 0$.

The value of t_2 is defined as:

$$t_2 = \min \{\varepsilon \frac{D(j)}{|D_\varepsilon(j)|} ; D(i) \geq 0; D_\varepsilon(i) < 0\} = \frac{D(j_2)}{D_\varepsilon(j_2)}$$

Let $t = \min(t_1, t_2)$. Then: (4.16.)

(i) If $t = t_1 \leq t_2$ we unlabel process node j_1 and resource node i_1

(ii) If $t = t_2$ we label process node j_2 and unlabel process node j_1.

In the case where t_1 is not uniquely defined from equation 4.15 i_1 is chosen arbitrarily. In the same way, if t_2 is not uniquely defined from equation 4.16, the node j_2 is chosen arbitrarily. If $t = \infty$, then the same is true as in type (b) labeling.

Example 4.3. Figure 4-4 shows the RPM network with three resource and four process variables.

The current solution is feasible but not optimum.

The value of the variable for the third resource is negative, $Y(3) = -.379$.

We have:

$$I = \{1, 2, 3\};\ J = \{1, 2, 3, 4\},\ I^* = \{2, 3\},\ J^* = \{3, 4\}$$

(i) For the candidate for unlabeling resource node $i_1 = 3$ let $S(3) = \varepsilon$. Equation 4.9 and 4.10 become:

$$-7\ X_\varepsilon(3) + 8\ X_\varepsilon(4) = -\varepsilon$$

$$X_\varepsilon(3) + 3\ X_\varepsilon(4) = 0$$

The perturbated solution is:

$$X_\varepsilon(3) = .1\varepsilon;\ X_\varepsilon(4) = -.034\ \varepsilon$$

The value of t_1 from equation 4.11 is found to be:

$$t_1 = \min\left\{\varepsilon\ \frac{2.41}{|-.034|\varepsilon}\right\} = 70 \text{ therefore } j_1 = 4$$

(ii) The change in the residue $S(1)$ is:

$$S_\varepsilon(1) = 9 \cdot (.1) - 6 \cdot (-.034) = 1.1 > 0$$

Therefore $t_2 = \infty$ and $t_1 = \min\{70, \infty\}$. In this case unlabel resource node $i_1 = 3$ and process node $j_1 = 4$.

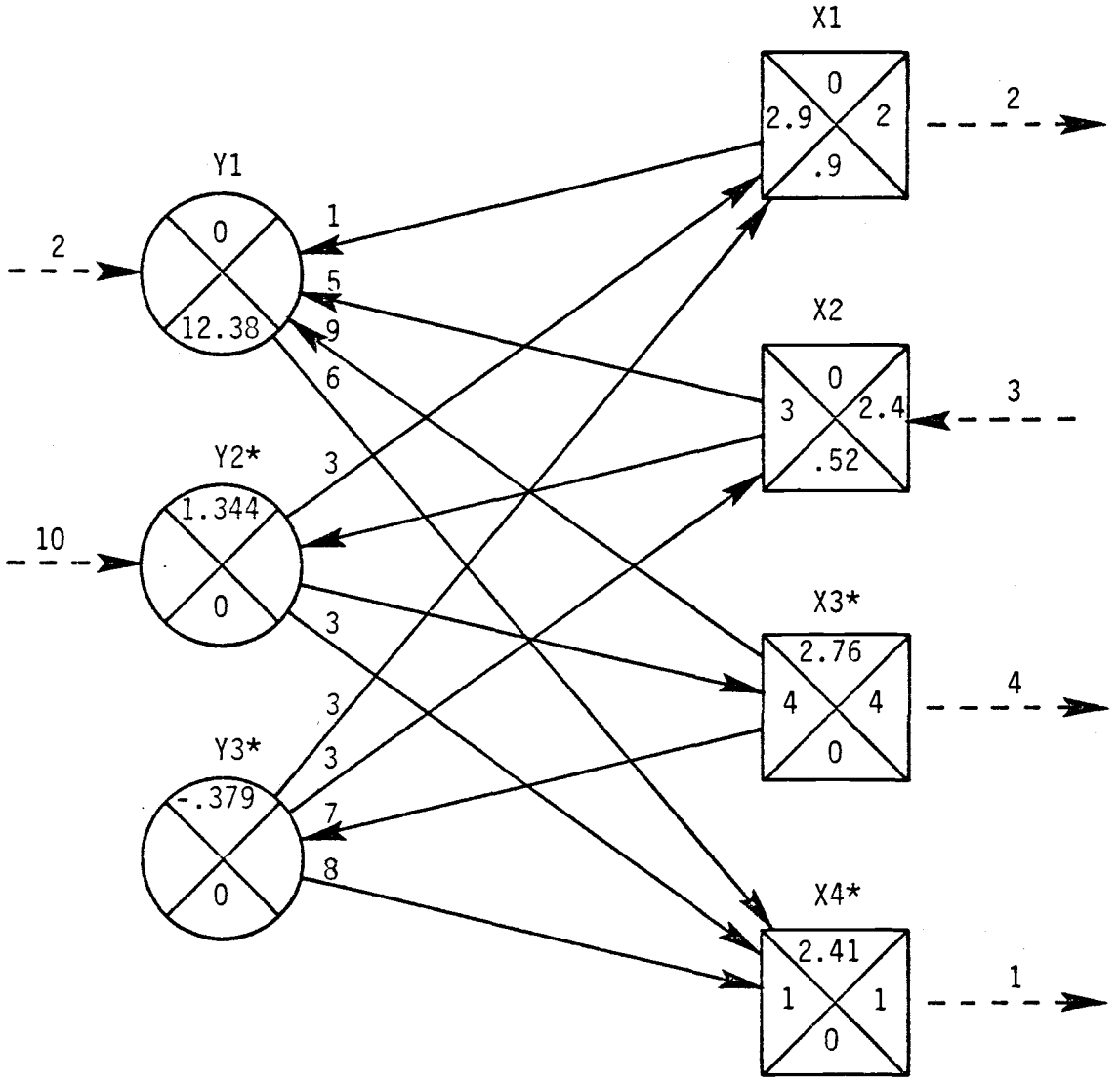


Figure 4-4. An Example for Type (c) Labeling

Termination Conditions

The new algorithm iterates through basic solutions working towards optimality (primal) or towards feasibility (dual)[3]. The final network solution is feasible if $S(i) \geq 0$ for all $i \in I$ and $X(j) \geq 0$ for all $j \in J$. It is optimal if $D(j) \geq 0$ for all $j \in J$ and $Y(i) \geq 0$ for all $i \in I$.

At each iteration the labeling process alters the sets of labeled nodes in such a way that the value of the objective function increases (decreases), provided that a change in the basic sets of nodes can be made, i.e. the value of the parameter t is finite ($t < \infty$). If $t = \infty$, then the next candidate node is considered.

For the final solution one of the following is possible:

1. The solution is primal and dual feasible (both the feasibility and optimality conditions are met). In this case the problem has an optimum solution.

2. The solution is feasible (i.e. the feasibility conditions are met) and there is at least one node with a negative value of dual variable (Y(i) or D(j)) with a corresponding value of t equal to ∞. In this case the primal problem is unbounded.

3. The solution is dual feasible (i.e. the optimality conditions for the primal are met), and there is at least one node with a negative primal variable (X(j) or S(i)) and a corresponding value of t equal

---

3. In certain literature, the term "optimal" is used to imply a feasible and optimal solution. In this dissertation, as in Wolfe (1973, pp. 110-111) and other literature, we consider a solution to be optimal, or dual feasible, even if it is not primal feasible.

to ∞. In this case the primal problem is nonfeasible (the dual problem is unbounded).

4. The solution is neither primal nor dual feasible and there is at least one primal variable value (X(j) or S(i)) and at least one dual variable value (Y(i) or D(j)) for which the corresponding values of ts are infinite. In this case the problem is neither feasible nor optimal (both primal and dual problems are unbounded).

Interpretation of Basis in RPM

In this chapter, the sets of labeled resource and process nodes I* and J*, at each iteration identified the critical constraints and the corresponding basic structural variables. The set of arcs of the original network that connect the labeled nodes forms a $\rho x \rho$ constraint matrix which corresponds to the basis matrix for the new algorithm. In the traditional simplex algorithm, and for the same basic solution, the set of basic variables(primal problem) would be the set consisting of the labeled process variables and unlabeled resource residues (slack variables) and the dimension of the basis is mxm.

V. FACTORIZATION

The purpose of this chapter is to find a method to decompose the labeled network into a triangular structure so that the process and resource variables can be easily evaluated. The labeling procedure, described in Chapter Four, guarantees that the number of labeled resource nodes, ρ, is equal to the number of labeled process nodes. The problem of finding values for the primal and dual variables is therefore, equivalent to the problem of finding a solution to a system of ρ linear equations in ρ unknowns.

Let $A = (a_{ij})$, a $\rho x \rho$ matrix. The matrix equation:

$Ax = b$, where $x^T = (x_1,\ldots,x_n)$, $b^T = (b_1,\ldots,b_n)$

defines a system of ρ linear simultaneous equations in ρ unknowns. By A_b we denote the $\rho x (\rho + 1)$ matrix with the vector b placed in the last column. Let $r(A)$ be the rank of matrix A. The following are true: (Ralston A, 1965)

(i) The system of equation $Ax = b$ has a solution if and only if:

$r(A) = r(A_b)$

(ii) If $r(A) = r(A_b) = \rho$, then there is a unique solution.

There are several methods for solving a linear system of equations divided mainly into two categories: direct methods and approximation methods. Among the direct methods, the method of triangular decomposition will be discussed in this chapter.

Theoretical Background

The best known and most widely used method for solving linear systems of equations is due to Gauss. The method is called "Gaussian Elimination" and it is the elementary procedure in which the first equation is used to eliminate the first variable from the remaining equations, the second equation is used next to eliminate the second variable from the remaining equations, etc. If the number of equations is ρ, then $\rho-1$ such eliminations can be performed and reduce the original system into a system with triangular structure. The solution of the system can be easily obtained. The new last equation gives the value of the last variable $x\rho$ of the system. This value is then substituted in the remaining new equations, and the value of the variable $x_{\rho-1}$ is obtained from the $\rho-1$ equation of the new system. After ρ steps, the values of the ρ variables of the system are explicitly evaluated. The method of finding the values of the variables from a triangular structured system is called "back-substitution." In general, the $\rho-1$ eliminations are not possible, unless certain row or column interchanges are performed during the elimination process.

The general procedure is described next analytically: Consider the system $Ax = b$ and set $A^{(1)} = A$, $b^{(1)} = b$. We select an arbitrary non-zero element $a_{i_1 j_1}$, and call it "the 1st pivot element." Using this pivot element, we can eliminate the variable x_{j_1} from all remaining equations. The variable x_{j_1} appears only in equation i_1. To eliminate the variable x_{j_1} from the kth equation ($k = 1, 2, \ldots, \rho$;

$k \neq i_1$) we use the multiplier $a_{kj_1}^{(1)} / a_{i_1 j_1}(1)$ to multiply equation i_1 and then subtract equation k_1 from it. The reduced system is written as: $A^{(2)}x = b^{(2)}$. $A^{(2)}$ is obtained from $A^{(1)}$ after eliminating variable x_{j_1} and deleting row i_1 and column j_1 of the original array $A^{(1)}$. The new system $A^{(2)}x = b^{(2)}$ is one of having ρ-1 equations in ρ-1 unknowns. The same elimination procedure yields the subsystems $A^{(k)}x = b^{(k)}$; $k = 1, 2, \ldots, \rho$, from which the variables $x_{j_1}, x_{j_2}, \ldots, x_{j_{k-1}}$ have been eliminated. Isaacson and Keller (1966) give a proof to the following theorem:

Theorem: Let the matrix A have rank r. Then we can find a sequence of distinct row and column indices $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$ such that the corresponding pivot elements in $A^{(1)}, A^{(2)}, \ldots, A^{(r)}$ are non-zero and $a_{ij}^{(r)} = 0$ if $i \neq i_1, i_2, \ldots, i_r$. Let us define the permutation matrices, whose columns are unit vectors.

$$P = \{e^{i_1}, e^{i_2}, \ldots, e^{i_r}, \ldots, e^{i_\rho}\}$$

$$Q = \{e^{j_1}, e^{j_2}, \ldots, e^{j_r}, \ldots, e^{j_\rho}\}$$ where i_k, j_k are pivotal indices and $\{i_k\}$ and $\{j_k\}$ are permutations of $1, 2, \ldots, \rho$. Then the system $By = g$, where $B = P^T AQ$, $y = Q^T x$, $g = P^T b$ is equivalent to $Ax = b$ and can be reduced to triangular form using Gaussian elimination with the natural order of pivots $(1,1), (2,2), \ldots, (r,r)$.

From the above theorem it is clear that when $\rho(A) = \rho$, A can be factorized as $A = LU$, where L is a lower triangular matrix, in which the diagonal elements are ones.

When the matrix A has been decomposed into the product LU, the

solution of $Ax = b$ can be found by first solving the system $Ly = b$ and then solving $Ux = y$, since $Ax = LUx = Ly = b$.

In general, there are many ways of choosing the pivots (i_k, j_k) during the elimination process, depending on the accuracy required and the way the array A is stored.

From the above discussion, it is clear that when the labeling process defines a set of resource nodes and a set of process nodes that form a nonsingular array A, then the labeled network can be transformed into a triangular structure, i.e. it can be decomposed into a lower triangular array L and an upper triangular array U. An analytic procedure to evaluate the elements of L and U is given next. To simplify the notation, position for the pivot elements is ignored (in the subsequent section of this chapter, the choice of the pivot elements, and the resulting permutations will be discussed in detail).

Suppose next that array A can be decomposed into the product LU, where $A = (a_{ij})$, $L = (l_{ij})$ and $U = (u_{ij})$. (Figure 5-1)

For the first row of L and the first column of U, the decomposition gives:

$$u_{11} = a_{11},\ l_{11} = 1$$

For the second row of L and the second column of U, we calculate:

$$u_{12} = a_{12},\ l_{21} = a_{21}/u_{11} \text{ and}$$

$$u_{22} = a_{22} - l_{21} \cdot u_{12},\ l_{22} = 1$$

For any subsequent row k; $k = 3, \ldots, \rho$ of L and the corresponding

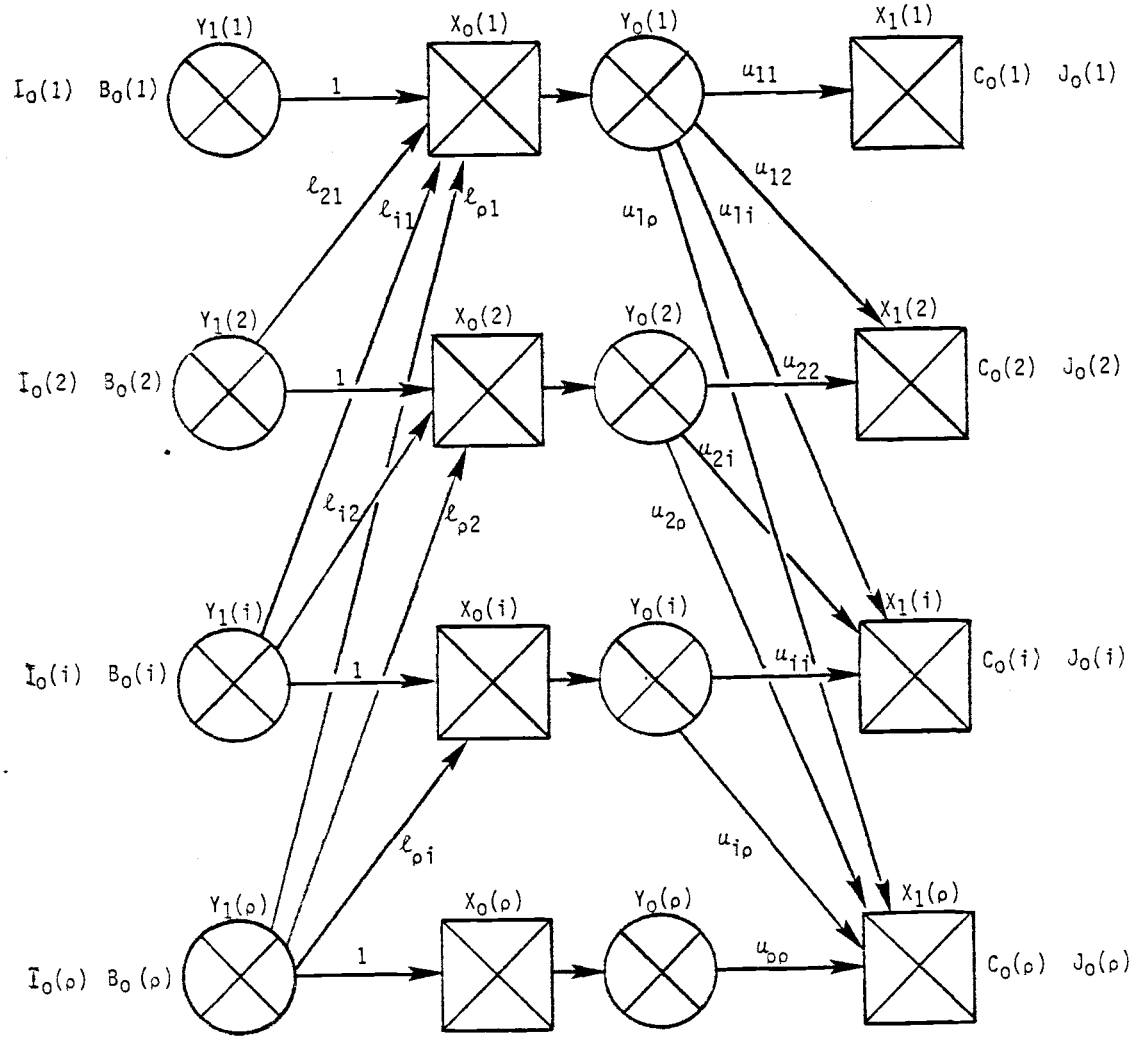


Figure 5-1. The Basis Factorization Scheme.

kth column of U, the analytic formulas are: (Ralston, A., 1965)

$$u_{kj} = a_{kj} - \sum_{l=1}^{k-1} l_{kl}u_{lj} \text{ for } j > k \text{ and}$$

$$l_{ik} = \frac{1}{u_{kk}} (a_{ik} - \sum_{l=1}^{k-1} l_{il}u_{lk}) \text{ for } i > k, \text{ and } l_{kk} = 1 \quad .$$

The Decomposition Method

In the remainder of this chapter, the method of decomposing the labeled network is described. Given a set of ρ labeled resource nodes and a set of ρ labeled process nodes, a decomposition scheme is derived for one of the following cases:

(i) The labeled network is decomposed, using the process graph.

The method is divided into ρ stages in a way that at each stage only one labeled process cluster is added to the decomposed network. At the end of the ρth stage a complete decomposition is obtained.

(ii) The labeled network is decomposed, using the resource graph.

The method is divided as in (i) into ρ stages such that only a labeled resource cluster is added to the decomposed network. Again, at the end of the ρth stage, a complete decomposition is obtained.

The Labeled Process Graph Decomposition

In the following discussion, Io contains the index number of the labeled resource nodes, corresponding to the rows of L, and Jo

contains the index number of process nodes corresponding to the rows of U.

The vector Bo is used to store the values of the arcs connecting each labeled process node and the labeled resource nodes.

The method is divided into ρ stages (where ρ is the number of labeled process nodes). At each stage r: $r = 1, 2, \ldots, \rho$, the elements of the first r columns of L and U are explicitly evaluated.

Stage 1. First, the labeled process node j_1, corresponding to the node number Jo (1), is considered. If $V_{j_1} = \{(h, v_h);\ h \in J_{j_1}\}$, then the value of the arcs v_h connecting the process node i_1 and the labeled resource nodes are entered into a ρ dimension vector Bo, where ρ is the number of labeled process nodes i.e. Bo (j) = v_h, if there is an h such that, h = Io (j), otherwise Bo (j) = 0; $j = 1, 2, \ldots, \rho$.

Let p_1 be the first non-zero element in Bo (i); $i = 1, 2, \ldots, \rho$.

If $p_1 = 1$ (i.e. if Bo (1) $\neq$ 0), then Bo (1) is used as a pivot element, and no resource node exchange is necessary. However, if $p_1 > 1$ then we exchange the following elements:

(i) Io (1) and Io (p_1)

(ii) Bo (1) and Bo (p_1)

Thus, if $p_1 > 1$ then, resource nodes 1 and p_1 are exchanged.

Next, the elements of the first column of L and U are calculated from:

$$u_{11} = \text{Bo (1)}$$

$$l_{i1} = \text{Bo (i)/Bo (1)};\ i = 2, 3, \ldots, \rho.$$

Stage 2. Let j_2 be the process node corresponding to Jo (2). Each of the arcs connecting the process node j_2 and the labeled resource nodes are entered into the vector Bo, as in stage one, i.e. if $V_{j_2} = \{(h, v_h); h \in J_{j_2}\}$, then: Bo (j) = v_h, if there is an h, such that h = Io (j), (otherwise Bo (j) = 0), j = 1, 2,..., ρ.

Next, update Bo (i): Bo (i) = Bo (i) - l_{i_1} Bo (1); i = 2, 3,..., ρ, and take u_{12} = Bo (1). At this point we proceed to find a new pivot element. First, a non-zero element is found: Let p_2 be the first non-zero element in Bo (i); i = 2, 3,..., ρ. If p_2 = 2, then no resource node exchange is necessary. However, if $p_2 > 2$ the resource nodes 2 and p_2 are exchanged, i.e.:

(i) Io (2) and Io (p_2) are exchanged (update index).

(ii) Bo (2) and Bo (p_1) are exchanged (switch the corresponding arcs.

(iii) l_{21} and l_{p_2} are exchanged (exchange rows #2 and p_2 of L).

The new pivot element is Bo (2). The remaining elements of the second column of U and L can be found:

$$u_{22} = \text{Bo (2) and}$$

$$l_{i_2} = \text{Bo (i)/Bo (2); i} = 3,\ldots, \rho$$

The process can be generalized for any given stage r; r = 3, 4, ..., ρ. For such an r, the corresponding stage is:

Stage r.

Step 1. For the process node j_r corresponding to Jo (r), the set of arcs connecting the process node j_r and the resource nodes, V_{j_r}, is used to enter the values of the arcs into the vector Bo, i.e.:

Let $V_{j_r} = \{(h, v_h); h \in J_{j_r}\}$ and $Bo(i) = v_h$, if there is an h, such that $h = Io(i)$, otherwise $Bo(i) = 0$; $i = 2, 3, \ldots, \rho$.

Step 2. We update Bo and calculate the first r-1 elements of the rth column of U, i.e.: for $j = 1, \ldots, r-1$ repeat (i) and (ii).

(i) $u_{jr} = Bo(j)$

(ii) update Bo (i): $Bo(i) = Bo(i) - l_{ij} Bo(j)$; $i = j + 1, \ldots, \rho$

Step 3. The next pivot element is selected. Let p_r be the position of the first nonzero element in Bo (i); $i = r, r + 1, \ldots, \rho$.

If $p_r = r$, the element Bo (r) can be used as a pivot (no resource node exchange is necessary). However, if $p_r > r$, then the resource nodes r and p_r are exchanged, i.e.:

(i) Exchange $Io(r)$ and $Io(p_r)$

(ii) Exchange $Bo(r)$ and $Bo(p_r)$

(iii) Exchange l_{rj} and $l_{p_r j}$; $j = 1, 2, \ldots, r$

Step 4. Calculate the remaining elements for the rth column of U and L. We have:

$$u_{rr} = Bo(r), \text{ and}$$

$$l_{ir} = Bo(i)/Bo(r); \quad i = r + 1, \ldots, \rho$$

Example 5.1. (Process entry)

Consider the RPM network in Figure (5.2a.). There are three resource and three process nodes. $I = \{1, 2, 3\}$, $J = \{1, 2. 3\}$. Suppose that all nodes are labeled. Let $Io(i) = I(i)$; $i = 1, 2, 3$.

Stage 1. For the process node $j_1 = 1$, we have:

$$V_1 = \{(1,2), (2,4), (3,1)\}$$

Let $Jo(1) = 1$. Each of the arcs connecting the process node (1),

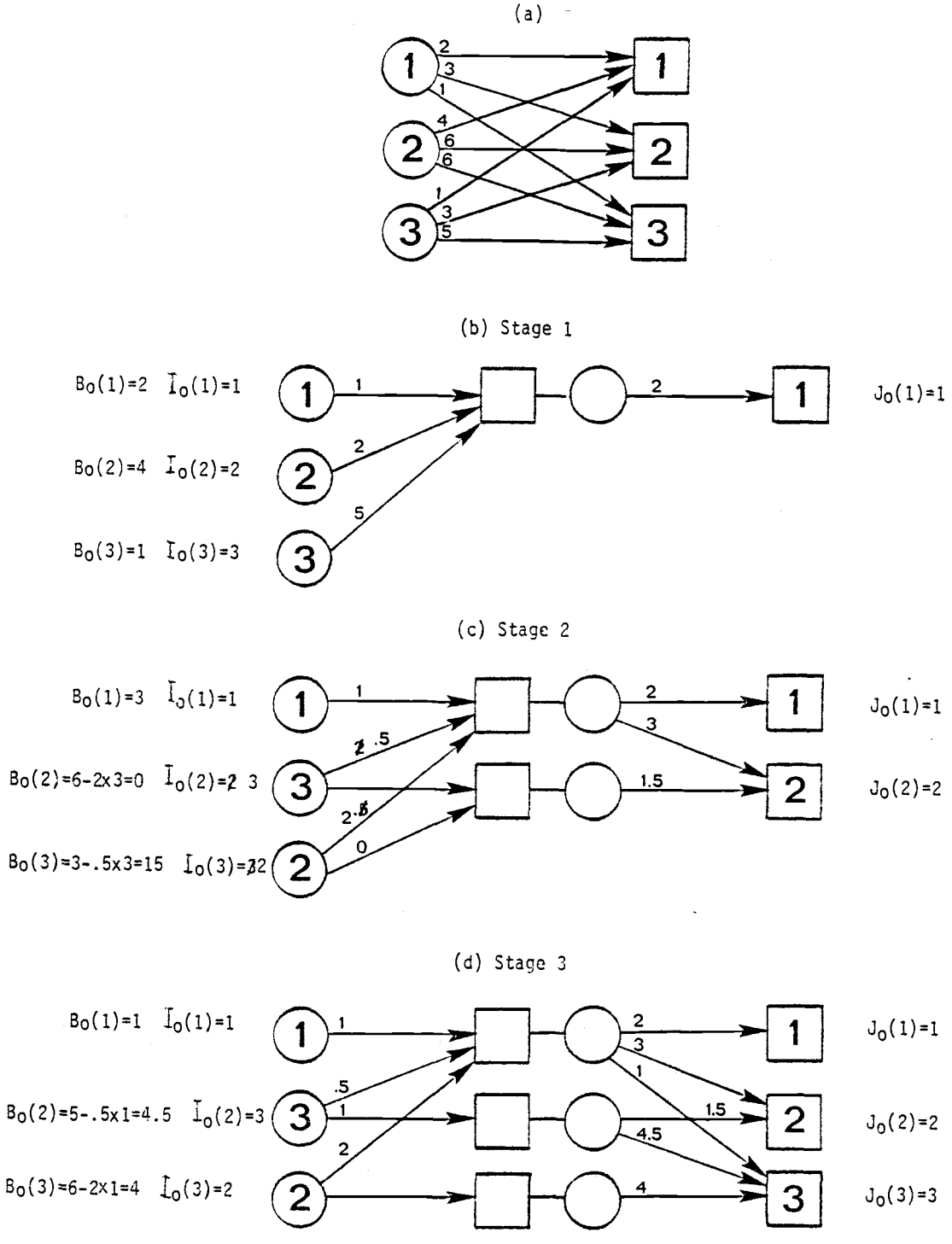


Figure 5-2. The Labeled Process Graph Decomposition.

and the resource nodes Io (i); i = 1, 2, 3, is entered in the bector Bo, such that

$$Bo\ (1) = 2\ (\text{i.e. } Bo\ (1) = v_1;\ 1 = Io\ (1))$$

$$Bo\ (2) = 4\ (\text{i.e. } Bo\ (2) = v_2;\ 2 = Io\ (2))$$

$$Bo\ (3) = 1\ (\text{i.e. } Bo\ (3) = v_3;\ 3 = Io\ (3))$$

Since Bo (1) = $2 \neq 0$ (i.e. $p_1 = 1$) no resource exchange is necessary. We compute:

$$u_{11} = Bo\ (1) = 2 \text{ and}$$

$$l_{21} = Bo\ (2)/u_{11} = 4/2 = 2;\ l_{31} = Bo\ (3) \neq u_{11} = 1/2 = .5$$

(Figure 5-2b)

Stage 2. For the process node $j_2 = 2$, we have:

$$V_2 = \{(1,3),\ (2,6),\ (3,3)\}$$

Let Jo (2) = 2. Each of the arcs connecting the process node (2) and the resource nodes Io (i); i = 1, 2, 3, i.e. entered in the vector Bo, as in stage 1. Thus, Bo becomes:

Bo (1) = 3

Bo (2) = 6

Bo (3) = 3

and, $u_{12} = Bo\ (1) = 3$. The values Bo (i); i = 2, 3 are updated:

$$Bo\ (2) = Bo\ (2) - l_{21} \cdot u_{12} = 6 - 2 \cdot 3 = 0$$

$$Bo\ (3) = Bo\ (3) - l_{31} \cdot u_{12} = 3 - .5 \cdot 3 = 1.5$$

In this case Bo (2) is zero. The first nonzero element for Bo (i); i = 2, 3, is Bo (3) = 1.5. Here, the interchange of resource nodes two and three is necessary. Set Bo (2) = 3, Bo (3) = 6 and $l_{21} = .5$, $l_{31} = 2$. Next, we take $u_{22} = Bo\ (3) = 1.5$, and

$l_{32} = [Bo\ (3) - l_{31}u_{12}]/u_{22} = (6 - 2 \cdot 3)/1.5 = 0.$ (Figure 5-2c.)

Stage 3. Process node $j_3 = 3$, is next entered for factorization. For this process node (3), we have:

$$V_3 = \{(1,1),\ (2,6),\ (3,5)\}$$

The values Bo (i) are:

$$Bo\ (1) = 1,\ Bo\ (2) = 5,\ Bo\ (3) = 6 \text{ and } u_{13} = Bo\ (1) = 1$$

Updating these values we get:

$$Bo\ (2) = Bo\ (2) - l_{21} \cdot u_{13} = 5 - .5 \cdot 1 = 4.5 \text{ or } u_{23} = 4.5$$

$$Bo\ (3) = Bo\ (3) - l_{3i}u_{13} - l_{32}u_{23} = 6 - 2 \cdot 1 = 4 \text{ or } u_{33} = 4$$

(Figure 5-2d.)

The Labeled Resource Graph Decomposition

As in the previous case (process graph decomposition), we assume that Io contains the index numbers for the labeled resource nodes, corresponding to the rows of L, and Io contains the index number of the process nodes corresponding to the rows of U. The vector Co is used to store the values of the arcs, which connect each labeled resource node and the labeled process nodes.

At each stage r; $r = 1, 2, \ldots, \rho$, the elements of the first r rows of L and the first r rows of U are evaluated.

Stage 1. First, the labeled resource node i_1, corresponding to the node number Io (1), is considered. If $U_{i_1} = \{(k, u_k);\ k \in I_{i_1}\}$, then the values of the arcs connecting the resource node i_1 and the labeled process nodes are entered into the vector Co of ρ dimension, where ρ is the number of labeled process nodes, i.e.: $Co\ (i) = u_k$,

if there is a k such that, k = Jo (i), otherwise Co (i) = 0 where i = 1, 2,..., ρ.

Let p_1 be the first nonzero element is Co (i); i = 1, 2,..., ρ. If p_1 = 1 (i.e. if Co (1) $\neq$ 0), then Co (1) is used as a pivot element, and no process node interchange is necessary. However, if p_1 > 0, then we interchange the following elements:

(i) Jo (1) and Jo (p_1)

(ii) Co (1) and Co (p_1)

Thus, if p_1 > 1 then process nodes 1 and p_1 are exchanged next, the elements of the first row of L and U are calculated:

$$l_{11} = 1; \; u_{ij} = Co\ (j); \; j = 1, 2,..., \rho.$$

Stage 2. Let i_2 be the resource node corresponding to Io (2). Each of the arcs connecting the resource node i_2 and the labeled process nodes, are entered into the vector Co, as in stage one. I.e.: if $U_{i_2} = \{(k, u_k); k \in I_{i_2}\}$, then Co (i) = u_k if there is a k, such that k = Jo (i), (otherwise, Co (i) = 0) i = 1, 2,..., ρ. Next, we update Co (j):

$$Co\ (j) = Co\ (j) - l_{21} \cdot u_{ij}; \; j = 2, 3,..., \rho, \text{ where } l_{21} = \frac{Co(1)}{u_{11}}$$

At this point we proceed with a new pivot element. First a nonzero element is found:

Let p_2 be the first nonzero element in Co (j); j = 2, 3,..., ρ. If p_2 = 2, then no process node interchange is necessary. However, if p_2 > 2 the process nodes 2 and p_2 are rotated, i.e.:

(i) Jo (2) and Jo (p_2) are interchanged.

(ii) Co (2) and Co (p_2) are interchanged.

(iii) u_{21} and $u_{p_2 1}$ are interchanged.

The new pivot element is Co (2). The second column of L and U are evaluated:

$$l_{22} = 1 \text{ and } u_{2j} = Co\ (j);\ j = 2, \ldots, \rho.$$

The above stages can be generalized for any number r; r = 3, 4, ..., ρ. Thus, for any r, we have:

Stage r.

Step 1. For the resource node i_r, corresponding to Io (r), the set of arcs connecting the resource node i_r and the process nodes U_{i_r}, is used to enter the values of the arcs into the vector Co, i.e.: Let $U_{i_r} = \{(k, u_k);\ k \in I_{i_r}\}$ and Co (j) = u_k, if there is a k, such that k = Jo (j), (otherwise CO (j) = o) where j = 1, 2,..., ρ.

Step 2. We update Co and calculate the first r-1 elements of the rth row of L, i.e.: For i = 1, 2,..., r-1, we repeat (i) and (ii)

(i) $l_{ri} = Co\ (i)/u_{i1}$

(ii) Update Co (j): $Co\ (j) = Co\ (j) - l_{ri} u_{ij};\ j = r + 1, \ldots, \rho$

Step 3. The next pivot element is selected: Let p_r be the position of the first nonzero element in Co (j); j = r, r + 1,..., ρ.

If $p_r = r$, the element Co (r) can be used as a pivot (no process node exchange is necessary). However, if $p_r > r$ then the process nodes r and p_r are exchanged, i.e.:

(i) Io (r) and Jo (p_r) are exchanged.

(ii) Co (r) and Co (p_r) are exchanged.

(iii) u_{rj} and $u_{p_r j}$; j = 1, 2,..., r are exchanged.

Step 4. Calculate the remaining elements of the rth row of L and V.

$$l_{rr} = 1 \text{ and } u_{rj} = Co(j);\ j = r, \ldots, \rho.$$

Example 5.2. (Resource entry)

Consider the RPM network, shown in Figure 5-3.(a). Suppose that all of the resource and process nodes are labeled, i.e.:

$$I = \{1, 2, 3\}, \text{ and } Io(i) = I(i);\ i = 1, 2, 3$$

$$J = \{1, 2, 3\} \text{ and } Jo(j) = J(j);\ j = 1, 2, 3.$$

Stage 1. For the first resource node $i_1 = 1$, we have $U_1 = \{(1,2), (2,3), (3,1)\}$. The values of the arcs are then entered into Co, i.e.:

$$Co(1) = 2$$

$$Co(2) = 3$$

$$Co(3) = 1$$

Since $Co(1) \neq 0$, we have $p_1 = 1$ (no process node exchange is needed). The elements of the first row of L and U are calculated:

$$l_{11} = 1 \text{ and } u_{1j} = Co(j) \text{ i.e. } u_{11} = 2,\ u_{12} = 3 \text{ and } u_{13} = 1$$

Stage 2. For the second resource node $i_2 = Io(2) = 2$, we have $U_2 = \{(1,4), (2,6), (3,6)\}$ and Co becomes:

$$Co(1) = 4$$

$$Co(2) = 6$$

$$Co(3) = 6$$

Next, we update Co:

$$Co(j) = Co(j) - l_{21}u_{1j}, \text{ where } l_{21} = \frac{Co(1)}{u_{11}} = \frac{4}{2} = 2;\ j = 2, 3.$$

Thus, $Co(2) = 6 - 2 \cdot 3 = 0$; $Co(3) = 6 - 2 \cdot 1 = 4$.

The new pivot element is found next: $p_2 = 3$ (since $Co(3) = 4 \neq 0$). Process nodes 2 and 3 are exchanged:

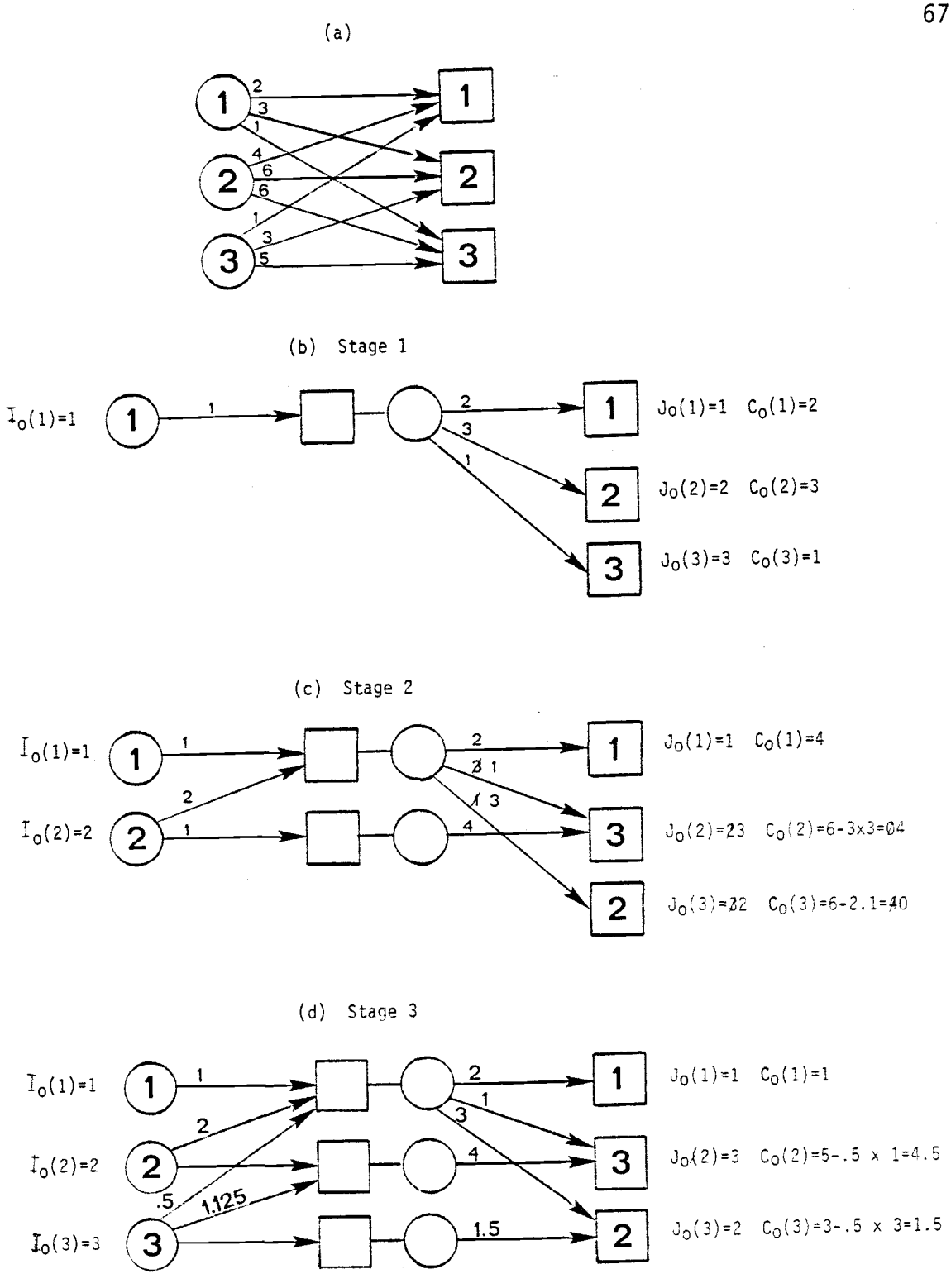


Figure 5-3. The Labeled Resource Graph Decomposition.

(i) Jo (2) = 3 and Jo (3) = 2

(ii) Co (2) = 4 and Co (3) = 0

(iii) $u_{12} = 1$ and $u_{13} = 3$

The second row of L and U are found from:

$l_{22} = 1$ and $u_{2j} = Co\ (j)$; $j = 2, 3$; i.e. $u_{22} = 4$ and $u_{23} = 0$

Stage 3.

Step 1. For the resource node i_3 Io (3) = 3, we have:

$U_3 = \{(1,1), (2,3), (3,5)\}$. The vector Co becomes:

Co (1) = 1

Co (2) = 5

Co (3) = 3

Step 2. We update Co and calculate the first 2 elements of the 3rd row of l, i.e.: For i = 1, 2 we repeat (i) and (ii)

(i) $l_{3i} = Co\ (i)/u_{i1}$ and

(ii) $Co\ (j) = Co\ (j) - l_{ri}u_{ij}$; $j = r + 1, \ldots, l$.

Let i = 1. Then:

(i) $l_{31} - 1/2 = .5$

(ii) for j = 2 we have Co (2) = 5 - .5 · 1 = 4.5

for j = 3 we have Co (3) = 3 - .5 · 3 = 1.5

Next, let i = 2. Then:

(i) $l_{32} = Co\ (2)/u_{21} = 4.5/4 = 1.125$

(ii) for j = 2 we have: Co (3) = 1.5 - 1.125 · 0 = 1.5

Step 3. $p_2 = 3$ (no process node exchange is needed).

Step 4. Compute $l_{33} = 1$ and $u_{37} = 1.5$.

Partial Factorization

In the last two sections, we considered the case where a numerical procedure was developed to factorize the labeled network by operating on the process or resource graph. However, at each iteration the factorization of the labeled network is required, when a factorization of a very similar labeled network has been obtained at the previous iteration. In order to reduce the number of operations required at each iteration, a technique which utilizes the additional information provided by the previous iteration, seems to be appropriate to apply.

The labeling process discussed in Chapter Four provides one of the following possible changes of the basis.

1. A new pair of nodes, one process node and one resource node per pair, is added to the labeled sets of nodes.

2. A newly labeled node (resource or process) replaces a previously labeled node (resource or process).

3. A pair of nodes (one resource and one process node) are deleted from the sets of labeled nodes.

Each of the above listed different possible changes in the basis are discussed in detail. For each one an economical (in the amount of computations needed for factorization) scheme will be developed.

Type (1) Factorization

In this section we consider the case in which a new pair of

nodes (one process node and one resource node), is added to the labeled sets of nodes.

Let Io be the set of the indexes for the labeled resource nodes, Jo the set of the indexes for the labeled process nodes, and ρ the number of elements in Io (the "set equality constraint" requires that Jo will have the same number of elements).

Suppose that the resource node i_1 and the process node j_1 are added to the labeled network, i.e. Io $(\rho + 1) = i_1$ and Jo $(\rho + 1) = j_1$. By adding a new pair of nodes, we add an extra row to the array L ($\rho + 1$ row), and a column to array U ($\rho + 1$ column), where L and U are the lower and the upper triangular matrices defined from the previous iteration.

The case where ρ is equal to zero is trivial since $l_{11} = 1$ and u_{11} is equal to the value of the arc connecting the resource and process nodes.

Let $\rho = 1$. Set Io $(2) = i_1$ and Jo $(2) = j_1$. We consider $V_{j_1} = \{(h, v_h);\ h \in J_{j_1}\}$ and for $j = 1, 2$ we set: Bo $(j) = v_h$, where $h =$ Io (j), i.e. Bo (1) is the value of the arc connecting the newly labeled process node j_1 and the resource Io (1), and Bo (2) is the arc connecting the new pair of labeled nodes.

Next, we consider $U_{i_1} = \{(k, u_k);\ k \in I_{i_1}\}$ and for $i = 1, 2$ we set Co $(i) = u_k$ where $k =$ Jo (i) ; i.e. Co (1) is the value of the arc connecting the resource node i_1 and the process node Jo (1), and Co (2) is the arc connecting the newly labeled pair of nodes i_1 and

j_1. Clearly Bo (2) = Co (2). (Figure 5.4)

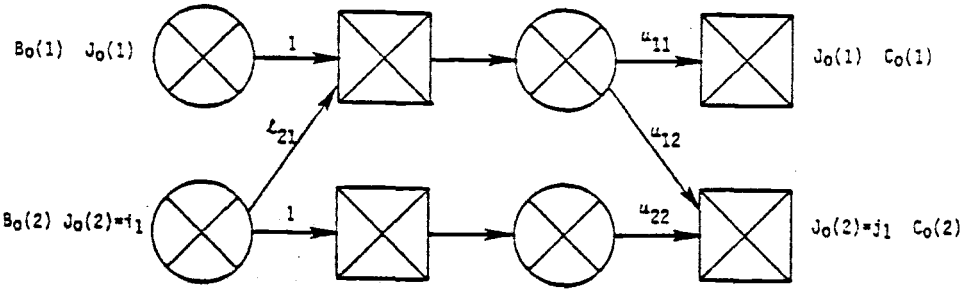


Figure 5.4. Adding a new pair of nodes to the basic nodes.

It follows that the new row of L is $l_{22} = 1$ and $l_{21} \cdot u_{11} = Co\ (1)$, i.e. $l_{21} = \dfrac{Co(1)}{u_{11}}$. For the new column of U we have:

$$u_{12} = Bo\ (1) \text{ and } u_{22} = Bo\ (2) - l_{21}u_{12}$$

In general, for $\rho \geq 2$ the analytic formula for computing the elements of the $\rho + 1$ row of L is:

$$l_{\rho+1,i} = \frac{1}{u_{ii}} \left[Co\ (i) - \sum_{k=1}^{i-1} l_{\rho+1,k}u_{ik}\right] \quad i = 1, 2, \ldots, \rho \text{ and } l_{\rho+1,p+1}=1$$

and the analytic formula for finding the elements of the $\rho+1$ column of U is:

$$u_{i,\rho+1} = Bo\ (i) - \sum_{u=1}^{i-1} l_{ik}u_{k\rho};\ i = 1, 2, \ldots, \rho+1$$

The above factorization is possible, provided that the element $u_{\rho+1,\ \rho+1}$ is different from zero. If $u_{\rho+1,\ \rho+1}$ is equal to zero then clearly the determinant of U is zero ($|U| = u_{11} \cdot u_{22} \ldots, u_{\rho+1,\rho+1}$). In this case, $|A| = |L| \cdot |U|$, i.e. the determinant of A is zero, and the rows (or columns) of A are not linearly independent. Therefore, provided that the new labeled resource (or process) node does not provide the labeled network with a set of arcs which form a constraint which is redundant, the type (1) factorization is always possible.

The necessary and sufficient condition for all types of factor-

ization is that the labeled network corresponds to a set of linearly independent constraints.

Type (2) Factorization

In this section, we consider the case in which the labeling process described in Chapter Four, provides with one of the following changes in the basis:

(a) A newly labeled process node replaces a previously labeled process node (process exchange), or

(b) A newly labeled resource node, replaces a previously labeled resource node (resource exchange).

Both (a) and (b) are described next in detail.

(1) Process Exchange

Suppose that a process node j_1 is to replace a process node j_2 in the basis at a given iteration and ρ is the number of labeled process nodes.

If Jo is the set of indexes for the labeled process nodes, there exists an r, $1 \leq r \leq \rho$, such that Jo (r) = j_2.

Step 1. Delete process node j_2 and add process node j_1. For $j = r, r+1, \ldots, \rho-1$ we update the set Jo, i.e. Jo (j) = Jo (j+1), thus, deleting the node j_2 from Jo. Next, we add the node j_1: Jo (ρ) = j_1. The node j_1 is placed at the last position of the set labeled process nodes.

Step 2. We continue with the process graph decomposition method described earlier in this chapter. Since at each stage k of the process graph decomposition method we calculate the kth column of

L and U arrays, it is clear that we can start the process at the r stage (r defined from Jo (r) = j_2)

(2) Resource Exchange

Suppose that at a certain iteration the resource node i_1 is to replace the resource node i_2 in the new basis. Since i_2 is in the basis, there is an r such that Io (r) = i_2 , where Io is the set of indexes for the labled resource nodes (Io (i); i = 1, 2,...,ρ).

Step 1. Delete resource node i_2 and add resource node i_1.
For i = r, r+1,..., ρ-1 we update Io; i.e. Io (j) = Io (j+1), thus deleting the node i_2. The resource node i_1 is added at the last position of the set Io, i.e. Io (ρ) = i_1.

Step 2. We continue with the resource graph decomposition method described earlier in this chapter. Recall that at each state of the method k, we calculate the kth row of L and U arrays. Clearly, the first r-1 rows of L and U (r defined from Io (r) = i_2) remain unchanged, and we start the method at stage r.

Type (3) Factorization

In this section we consider the case in which the labeling technique results with the following change in the set of basic (labeled) variables: A pair of nodes (a resource and process node) are deleted from the set of labeled nodes.

Let i_1 be the resource node, and j_1 be the process node, both to be deleted from the basis (unlabeled).

Let ρ be the number of labeled resource nodes. The necessary

steps, to update the factorization are:

Step 1. Delete process node j_1. Let r; $1 \leq r \leq \rho$ such that Jo (r) = j_1. Then for j = r, r+1,..., ρ-1 we replace Jo (j) with Jo (j+1), and take Jo (ρ) = 0. Thus, the process node j_1 is deleted from Jo.

Step 2. Delete resource node i_1. Again let r; $1 \leq r \leq \rho$ such that Io (r) = i_1. Then for i = r, r+1,..., ρ-1 we replace Io (i) with Io (i+1), and take Io (ρ) = 0. The resource node i_1 is deleted from Io.

Step 3. Apply the resource graph decomposition technique described earlier in this chapter for the set of labeled resource nodes Io(i); i = 1,..., ρ-1, and the set of labeled process nodes Jo (j); j = 1, 2,..., ρ-1.

Discussion

In this chapter, a general method for the triangular decomposition of the labeled network was discussed. The triangular basis can be obtained from the process graph (labeled process graph decomposition), by considering one labeled process node at each stage. The decomposition of the labeled network can also be obtained from the resource graph (labeled resource graph decomposition), by considering one resource node at each stage. In both cases, the decomposition is feasible, provided that the labeled network consists of a set of linearly independent constraints.

However, this is always the case. For example, in type (a) labeling, whenever j_1 replaces j_2 in the basis, the updated basis corresponds to a nonsingular array (using Cramer's rule and the fact that $X_{\varepsilon}(j_2) \neq 0$). Also, whenever j_1 and i_1 become labeled the new basis corresponds to a nonsingular array (using Gauss' method) The same is true for the other types of labeling.

During the decomposition process, the elements of a lower triangular array L of dimension $\rho x \rho$ (where ρ is the number of labeled resource nodes), and the elements of an upper triangular array U of the same dimension ($\rho x \rho$) are calculated. For storage purposes the elements of both arrays L and U can be stored in an array of dimension $\rho x \rho$.

At each iteration, the decomposition of the labeled network is required, when a decomposition of the labeled network of the previous iteration has been derived. The partial factorization methods described in this chapter can reduce the number of operations necessary for factorization at each iteration. The computational savings, in general, depend on the position of the exchanged nodes (resource or process) in the basis.

VI. BALANCING

Introduction

Balancing is the process of updating the values of the variables for the labeled nodes and the residues for the unlabeled nodes. Applying the complementary slackness theorem, the residues of the labeled nodes, and the variables of the unlabeled nodes are set equal to zero.

First, the primal and dual basic structural variables are obtained from the factorized basis by backsubstitution. Next, the values of the residues are computed individually for each resource or process node. (Resource or process nodes which are not connected to a basic node do not need balancing.)

Analytically, the balancing equations for the network are given below:

(1) Labeled nodes:

$$(XU)_i = B(i);\ i \in I^*$$

$$(YV)_j = C(j);\ j \in J^*$$

(2) Unlabeled nodes:

$$S(i) = B(i) - (XU)_i;\ i \in I^o$$

$$D(j) = (YV)_j - C(j);\ j \in J^o$$

The Process of Balancing

The labeled network, when it is factorized into the product of a lower triangular (L) and an upper triangular (U) matrix, provides

a simple procedure for computing the basic variables. This procedure is called backsubstitution, and it is used for finding the values of the primal and dual basic variables. This procedure is explained below.

The artificial process nodes Xo are used first to find a solution to the system of equations, LXo = Bo, where L is the lower triangular array and Bo the vector containing the constants $B_{(i)}$ corresponding to the labeled resource nodes. For the first variable Xo(1), we have Xo (1) = Bo (1). Substituting this value of Xo (1) in the second equation, we find Xo (2), etc. After a solution for Xo has been found, array U is used to calculate the process variables. In this case we solve the system UX_1 = Xo applying the same procedure, this time backwards, i.e. the value of the variable X_1 (ρ) is found from the last equation as X_1 (ρ) = $\frac{1}{u_{\rho\rho}}$ Xo (ρ), etc.

To calculate the values of the dual variables Y_1, we use the artificial nodes Yo as the solution of the system UYo = Co. Next, Yo is found by backsubstitution. The dual variables are computed next by solving the system LY_1 = Yo.

After the solution for X_1 and Y_1 has been found, the process and resource variables are identified from the set of indexes Io and Jo and the residue of each unlabeled node is updated. The different steps are:

Step 1. The process variables are computed for the labeled process nodes: (the indices Io, Jo are defined in Chapter Five).

For $i = 1, 2, .., \rho$; let $Bo(i) = B_{(k)}$ where $k = Io(i)$.

Let,

$$Xo(j) = Bo(j) - \sum_{k=1}^{j-1} l_{jk} Xo(k) \; ; \; j=1, 2, \ldots, \rho$$

and $$X_1(j) = \frac{1}{u_{jj}} [Xo(j) - \sum_{k=j}^{\rho-1} u_{jk} \cdot X_1(k)] \; j = \rho, \rho-1, \ldots, 1$$

The corresponding process variables are given by:

$$X(k) = X_1(j); \; k = Jo(j); \; j = 1, 2, \ldots, \rho.$$

Step 2. The resource variables are computed for the labeled resource nodes:

For $j = 1, 2, \ldots, \rho$; let $Co(j) = C_{(h)}$ where $h = Jo(j)$.

Let,

$$Yo(i) = \frac{1}{u_{ii}} \left[C(i) - \sum_{i=1}^{i-1} u_{ki} Yo(k) \right], \; i = 1, 2, \ldots, \rho$$

and $$Y_1(i) = Yo(i) - \sum_{k=1}^{\rho-1} l_{ki} Y_1(k); \; i = \rho, \rho-1, 1.$$

The resource variables are given by:

$$Y(h) = Y_1(i); \; h = Io(i), \; i = 1, 2, \ldots, \rho$$

Step 3. Update resource residues for unlabeled resource nodes:

$$S(i) = B(i) - \sum_{j} u_j X(j) \; ; \; i \in I_i^{\circ}; \; j \in I_i^{*}$$

Step 4. Update process residues for unlabeled process nodes:

$$D(j) = \sum_{i} v_i Y(i) + C(j); \; j \in J_j^{o}; \; i \in J_j^{*}$$

Illustrative Example 6.1.

Consider the RPM model shown in Figure 6-1. The network contains three resource nodes (Y_1, Y_2, and Y_3) and four process nodes (X_1, X_2, X_3, and X_4). The initial zero solution for the primal and dual variables is entered and the resource and process residues are updated. The zero solution contains negative residues and is neither feasible nor optimum: (S (2) = -4; S (3) = -10) (D (1) = -2; D (2) = -1, D (3) = -4, D (4) = -5.

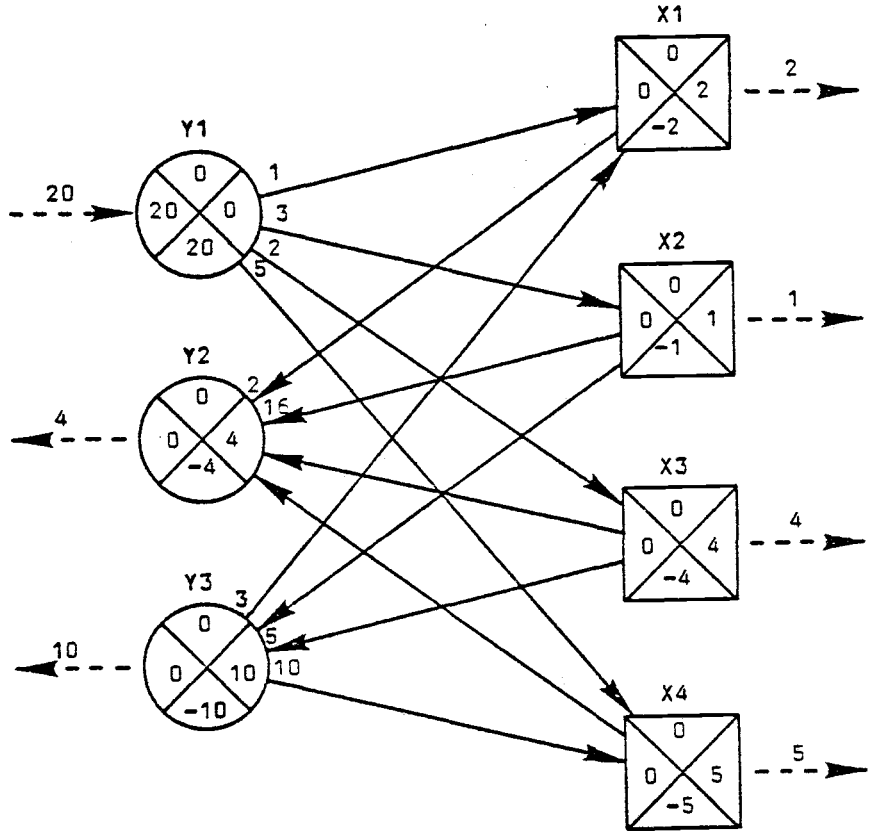
Iteration 1.

(C.1.) The residue of the first process X (1) is negative D(1) = -2, and the process is considered a candidate for labeling. Therefore, type (a) labeling is applied.

(L.1.) Let $j_1 = 1$ for choosing X(1) as a candidate. For this process node, $j_1 = 1$, we have:
$J_1 = \{1, 2, 3\}$; $J_1^* = \emptyset$, $J_1^\circ = \{1, 2, 3\}$, since the process node 1 is connected to resource nodes Y_1, Y_2, Y_3 which are all unlabeled.

(i) Let X (1) = ε. The values of the parameters $t_1 = t_2 = \infty$. Since J_1^* is empty, we have $t_1 = \infty$

(ii) To find the value of t_2, we calculate the changes of the residues when X(1) = ε for all resource nodes with non-negative residue. In this case $S_\varepsilon(1) = -1 \cdot \varepsilon$ and the value of t_2 is found from Equation 4.4 to be:
$t_2 = \min_1 \{\frac{20}{|-1|}\} = 20$ and the corresponding to



Linear RPM Networks

RPM test file (3X4)

Iteration No. 0 Objective Function 0.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	RES001		0.00	20.00	20.00	4	1.0(1)	3.0(2)	2.0(3)	5.0(4)
2	RES002		0.00	-4.00	-4.00	4	-2.0(1)	-16.0(2)	-1.0(3)	-1.0(4)
3	RES003		0.00	-10.00	-10.00	4	3.0(1)	-1.0(2)	-5.0(3)	10.0(4)

Process Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	PRO01		0.00	2.00	-2.00	3	1.0(1)	-2.0(2)	3.0(3)	
2	PRO02		0.00	1.00	-1.00	3	3.0(1)	-16.0(2)	-1.0(3)	
3	PRO03		0.00	4.00	-4.00	3	2.0(1)	-1.0(2)	-5.0(3)	
4	PRO04		0.00	5.00	-5.00	3	5.0(1)	-1.0(2)	10.0(3)	

Figure 6-1. The Initial RPM Network for Illustrative Example 6.1.

the minimum value resource node is $i_1 = 1$.

(iii) Find the value of t:

$$t = \min (t_1, t_2)$$

$$t = \min (20, \infty) = 20, \text{ i.e. } t = t_1$$

Label resource node 1 and process node 1.

(F.1.) At the first iteration $\rho = 1$. LU = A or (l_{11}) $(u_{11}) = 1$ since the elements of L and U for a one-dimension case are $l_{11} = 1$ and $u_{11} = 1$. Graphically, u_{11} = ar connecting $Y_{(1)}$ to $Xo_{(1)}$ and l_{11} = the new arrow from $Y_1(1)$ to Xo (1).

Next, the indexes of the labeled nodes are updated.

Io (1) = 1, Jo (1) = 1.

(B.1.) The vectors Bo and Co are:

Bo (1) = 20, Co (1) = 2

1. Update labeled network (Figure 6-2).

(a) Update process variables:

Xo (1) = 20, X1 (1) = 20 and X (1) = 20

(b) Update resource variables:

Yo (1) = 2, Y1 (1) = 2, and Y (1) = 2

2. Update unlabeled network.

(a) Update resource residues:

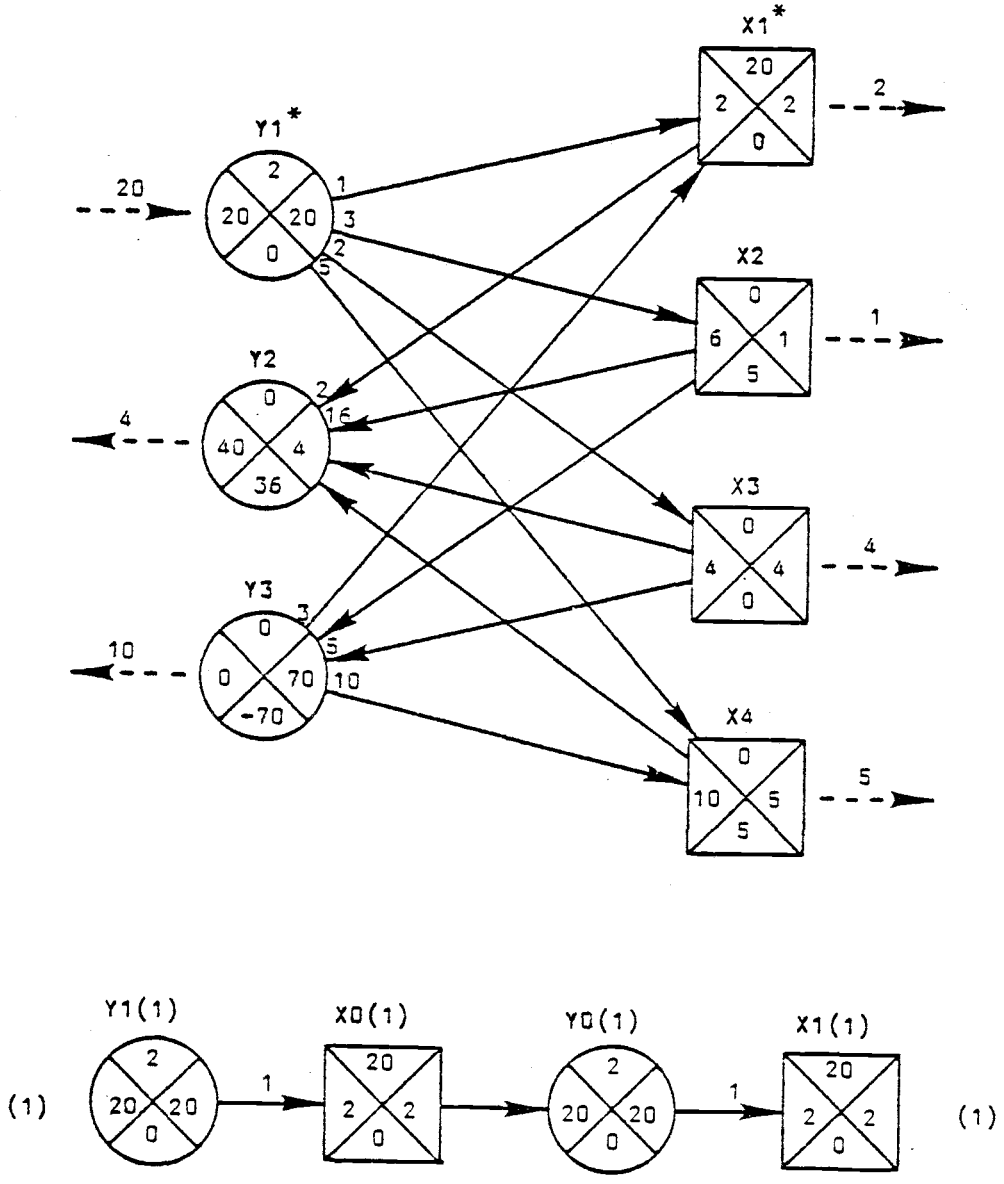


Figure 6-2. The First Iteration for Illustrative Example 6.1.

$$S(2) = -4 + 2 \cdot 20 = 36$$

$$S(3) = -10 - 3 \cdot 20 = -70$$

(b) Update process residues:

$D(2) = -1 + 2 \cdot 3 = 5$, $D(3) = -4 + 2 \cdot 2 = 0$, and

$$D(4) = -5 + 2 \cdot 5 = 5$$

Iteration 2.

First, the complementary slackness theorem applies: $S(1) = 0$ and $D(1) = 0$. The current solution is optimum (i.e. the optimality conditions $D(j) \geq 0$ and $Y(i) \geq 0$ are satisfied) but not feasible (i.e. the feasibility conditions $S(i) \geq 0$ and $X(j) \geq 0$ are not satisfied).

(C.2.) The third resource has negative residue. Type (b) labeling is applied.

(L.2.) (i) First the value of t_1 is found. Let $Y_\varepsilon(3) = \varepsilon$. Equation 4.5 becomes $(Y_\varepsilon V)_1 = 1 \cdot Y_\varepsilon(1) = -3 \cdot \varepsilon$ or $Y_\varepsilon(1) = -3 \cdot \varepsilon$. Therefore, $t_1 = \min\{\frac{2}{3}\} = \frac{2}{3}$; $i_2 = 1$

(ii) Next, the changes of the process residues when $Y_\varepsilon(3) = \varepsilon$ for all process nodes with non-negative residue are computed:

$$D(2) = 5 \text{ and } D_\varepsilon(2) = -3 \cdot 3 \cdot \varepsilon - \varepsilon = -10\varepsilon$$

$$D(3) = 0 \text{ and } D_\varepsilon(3) = -2 \cdot 3 \cdot \varepsilon - 5\varepsilon = -11\varepsilon$$

$$D(4) = 5 \text{ and } D_\varepsilon(4) = -5 \cdot 3 \cdot \varepsilon + 10\varepsilon = -5\varepsilon$$

The value of t_2 given from Equation 4.8 is:

$$t_2 = \min_{2,3,4} \left\{\frac{5}{10}, \frac{0}{11}, \frac{5}{5}\right\} = 0; \; j_1 = 3.$$

Therefore, we label $i_1 = 3$ and $j_1 = 3$

(F.2.) The dimension of the labeled network is $\rho = 2$.

First, we update Bo and Co:

For the process node 3 we have, Bo (1) = 2 and Bo (2) = -5, and for the resource node 3, Co (1) = 3 and Co (2) = -5.

The type (1) factorization gives:

$$l_{21} = 3 \text{ and } l_{22} = 1$$

$$q_{12} = 2 \text{ and } q_{22} = -5 - 3 \cdot 2 = -11$$

(B.2.) First, the vectors Bo and Co are updated:

For the resource nodes, Bo (1) = 20 and Bo (2) = -10, and for the process nodes, Co (1) = 2 and Co (2) = 4.

1. Update labeled network:

(a) Update process variables:

Xo (1) = 20

Xo (2) = -10 - 3 · 20 = -70

X1 (2) = (-1/11) · (-70) = 6.36, X (3) = 6.36

X1 (1) = 20 - 2 · 6.36 = 7.27, X (1) = 7.27

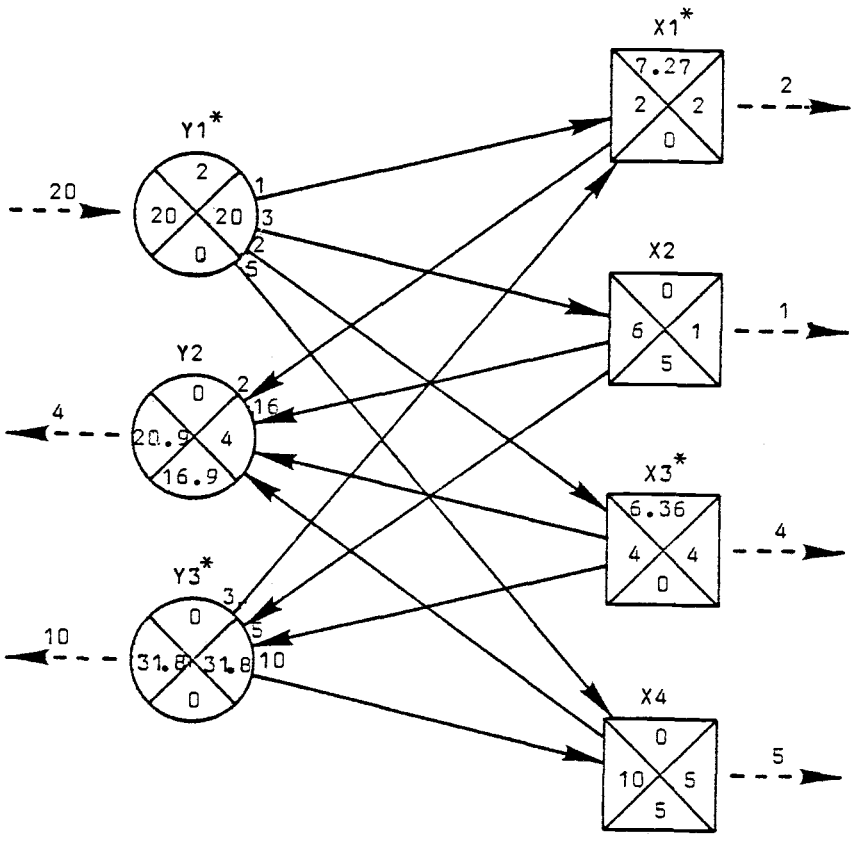
(b) Update resource variables:

Yo (1) = 2

Yo (2) = (-1/11) · (4 - 2 · 2) = 0

Y1 (2) = 0, Y (3) = 0

Y1 (1) = 2 - 3 · 0 = 2, Y (1) = 2



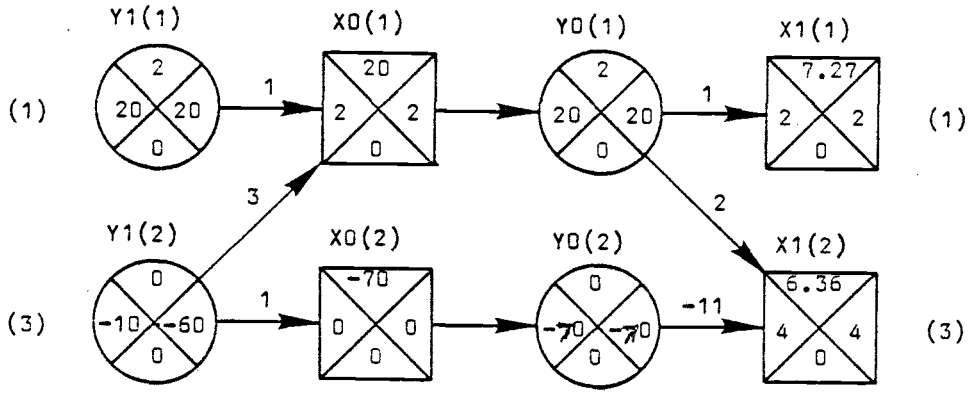


Figure 6-3. The Optimum Network for Illustrative Example 6.1.

2. Update unlabeled network:

(a) Update resource residues:

$$S(2) = -4 + 2 \cdot 7.27 + 6.36 = 16.9$$

(b) Update process residues:

$$D(2) = -1 + 2 \cdot 3 - 5 \cdot 0 = 5$$

$$D(4) = -5 + 5 \cdot 2 + 10 \cdot 0 = 5$$

The current solution is optimum. The value of the objective function is:

$$z = 2 \cdot 7.27 + 4 \cdot 6.36 = 40$$

Linear RPM Networks

RPM test file (3X4)

Iteration No. 1 Objective Function 40.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	RES001	*	2.00	20.00	0.00	4	1.0(1)	3.0(2)	2.0(3)	5.0(4)
2	RES002		0.00	-4.00	36.00	4	-2.0(1)	-16.0(2)	-1.0(3)	-1.0(4)
3	RES003		0.00	-10.00	-70.00	4	3.0(1)	-1.0(2)	-5.0(3)	10.0(4)

Process Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	PR001	*	20.00	2.00	0.00	3	1.0(1)	-2.0(2)	3.0(3)	
2	PR002		0.00	1.00	5.00	3	3.0(1)	-16.0(2)	-1.0(3)	
3	PR003		0.00	4.00	0.00	3	2.0(1)	-1.0(2)	-5.0(3)	
4	PR004		0.00	5.00	5.00	3	5.0(1)	-1.0(2)	10.0(3)	

Linear RPM Networks

RPM test file (3X4)

Iteration No. 2 Objective Function 40.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	RES001	*	2.00	20.00	0.00	4	1.0(1)	3.0(2)	2.0(3)	5.0(4)
2	RES002		0.00	-4.00	16.30	4	-2.0(1)	-16.0(2)	-1.0(3)	-1.0(4)
3	RES003	*	0.00	-10.00	0.00	4	3.0(1)	-1.0(2)	-5.0(3)	10.0(4)

Process Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Nd)	Arc(Nd)	Arc(Nd)	Arc(Nd)
1	PR001	*	7.27	2.00	0.00	3	1.0(1)	-2.0(2)	3.0(3)	
2	PR002		0.00	1.00	5.00	3	3.0(1)	-16.0(2)	-1.0(3)	
3	PR003	*	6.36	4.00	0.00	3	2.0(1)	-1.0(2)	-5.0(3)	
4	PR004		0.00	5.00	5.00	3	5.0(1)	-1.0(2)	10.0(3)	

Current Solution is Optimum

Figure 6-4. Different Tableau for Illustrative Example 6.1.

Illustrative Example 6.2.

Consider the RPM model shown in Figure 6-5. The network contains three resource and two process nodes. The initial zero solution for the primal and dual variables is entered and the resource and process residues are updated. The zero solution is feasible but not optimum.

Iteration 1. (Figure 6-5)

(C.1.) The residue of the first process is negative and the process is considered as a candidate for labeling. In this case, type (a) labeling is applied.

(L.1.) Let $j_1 = 1$. For this process node we have:

$J_1 = (1, 2, 3)$ and $V_1 = ((1,1), (2, 12), (3,5))$.

Also, $J_1^* = \emptyset$; $J_1^o = J_1$.

Let t_1, t_2, $t = \infty$

(i) Find the value of t_1:

Since J_1^* is empty, we have $t_1 = \infty$

(ii) Find the value of t_1:

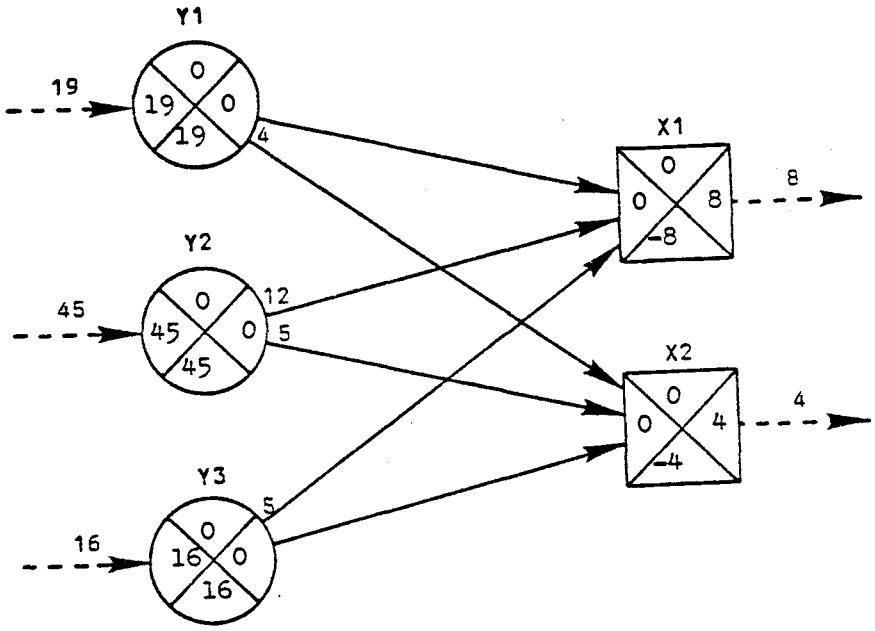
$t = \min (19/1, 45/12, 16/5) = 16/5$. Therefore, $i_1 = 3$.

(iii) Find the value of t:

$t = \min (16/5, \infty) = 16/5$; i.e. $t = t_1$.

Label resource node 3 and process node 1.

(F.1.) At the first iteration $\rho = 1$. The elements of L and U are $l_{11} = 1$ and $u_{11} = 5$.



Linear RPM Networks

RPM test file (3X2)

Iteration No. 0 Objective Function 0.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)
1	RES001		0.000	19.000	19.000	2	1.0 (1)	4.0 (2)	
2	RES002		0.000	45.000	45.000	2	12.0 (1)	5.0 (2)	
3	RES003		0.000	16.000	16.000	2	5.0 (1)	1.0 (2)	

Process Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)
1	PR001		0.000	8.000	-8.000	3	1.0 (1)	12.0 (2)	5.0 (3)
2	PR002		0.000	4.000	-4.000	3	4.0 (1)	5.0 (2)	1.0 (3)

Figure 6-5. The Initial RPM Network for Illustrative Example 6.2.

Next, the indexes of the labeled nodes are updated.

Io (1) = 3, Jo (1) = 1.

(B.1.) The vectors Bo and Co are: Bo (1) = 19, Co (1) = 8.

1. Update labeled network.

(a) Update process variables:

Xo (1) = 16, X1(1) = 16/5 = 3.2 and X (1) = 3.2

(b) Update resource variables:

Yo (1) = 8/5 = 1.6, Y1 (1) = 1.6 and Y (3) = 1.6

2. Update unlabeled network.

(a) Update resource residues:

S (1) = 19 - 3.2 = 15.8

S (2) = 45 - 12 · 3.2 = 6.6

(b) Update process residues:

D (2) = -4 + 1.6 = -2.4

Iteration 2. (Figure 6-6)

First, the complementary slackness theorem applies:

B (3) = 0 and D (1) = 0

(C.2.) The second process variable has negative residue.

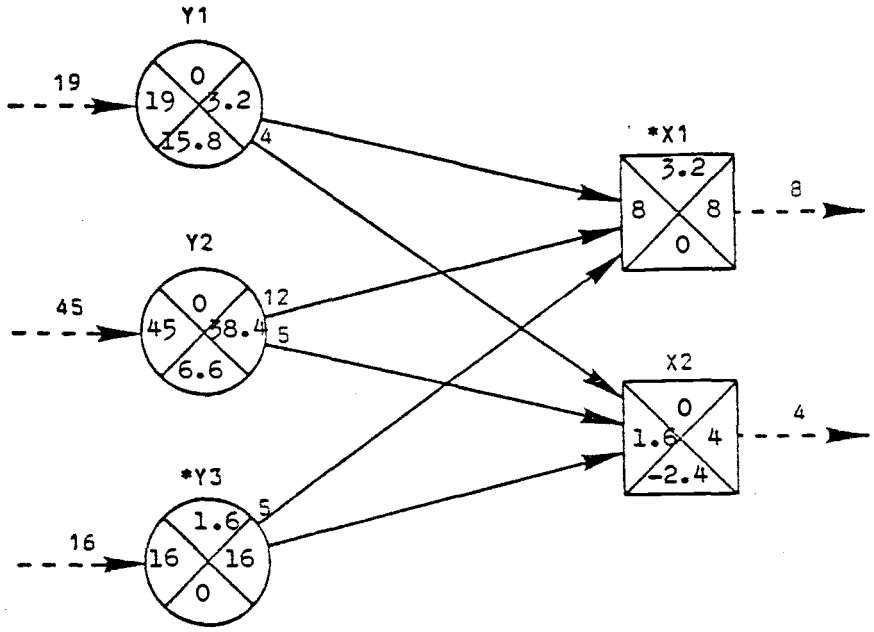
Type (a) labeling is applied.

(L.2.) Let $j_1 = 2$, we have $J_2 = (1, 2, 3)$

and $V_2 = ((1,4), (2,5), (3,1))$. Also,

$J_2^* = (3)$, $J_2 = (1,2)$. Let t_1, t_2, $t = \infty$

(i) Find the value of t_1: Let $X_\varepsilon(2) = \varepsilon$



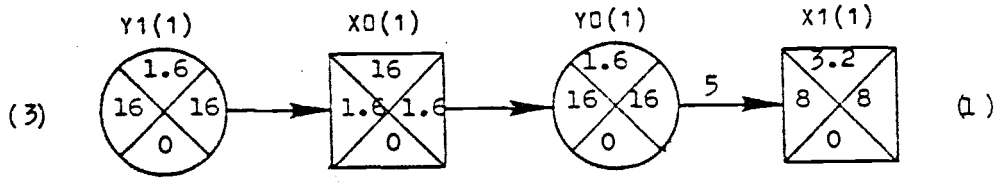


Figure 6-6. The First Iteration for Illustrative Example 6.2.

Then, 5 $X_\varepsilon(1) = -\varepsilon$ i.e. $X_\varepsilon(1) = -.2\varepsilon$

$t_1 = \min (3.2/.2) = 16$, and $j_2 = 1$

(ii) Next, the value of t_2 is computed. We have:

$S_\varepsilon(1) = +.2\varepsilon - 4\varepsilon = -3.8\varepsilon$; $S_\varepsilon(2) = 12 \cdot .2\varepsilon - 5\varepsilon = -2.6\varepsilon$

i.e. $t_2 = \min\{ \frac{15.4}{3.8} , \frac{6.6}{2.6} \} = \frac{6.6}{2.6} = 2.5$; $i_1 = 2$

(iii) Find the value of t:

$t = \min (2.5 , 16) = 2.5$. Therefore $t = t_2$

The resource node 2 and the process node 2 are labeled.

(F.2.) The dimension of the labeled network is, $\rho = 2$.

First, we update Bo and Co:

For the process node 2 we have, Bo (1) = 1 and Bo (2) = 5, and for the resource node 2, Co (1) = 12 and Co (2) = 5

The type (1) factorization gives:

$l_{21} = 12/5 = 2.4$ and $l_{22} = 1$

$u_{12} = 1$ and $u_{22} = 5 - 2.4 = 2.6$

(B.2.) First, the vectors Bo and Co are updated.

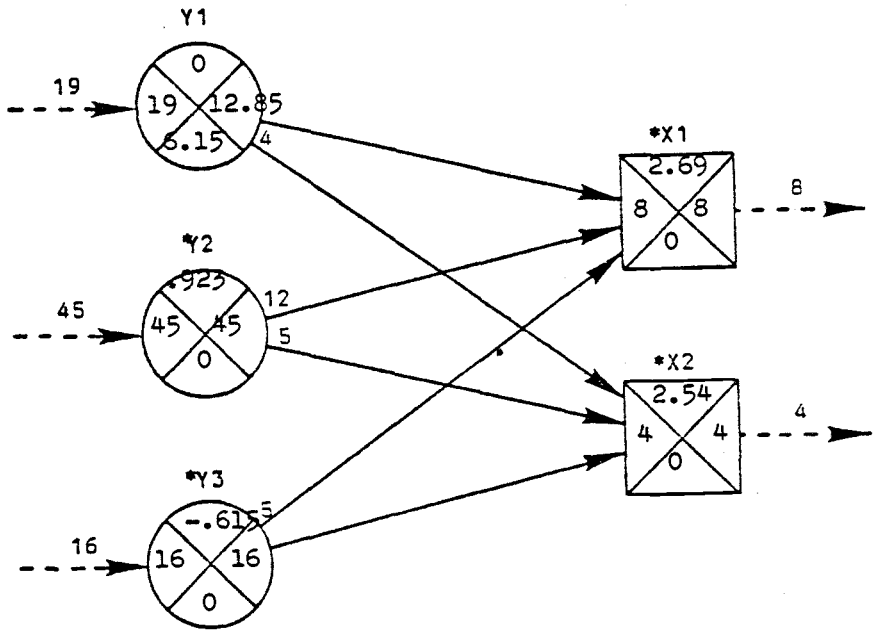
For the resource nodes, Bo (1) = 16 and Bo (2) = 45, and for the process nodes, Co (1) = 8 and Co (2) = 4

1. Update labeled network.

(a) Update process variables:

Xo (1) = 16

Xo (2) = 45 - 16 · 2.4 = 6.6



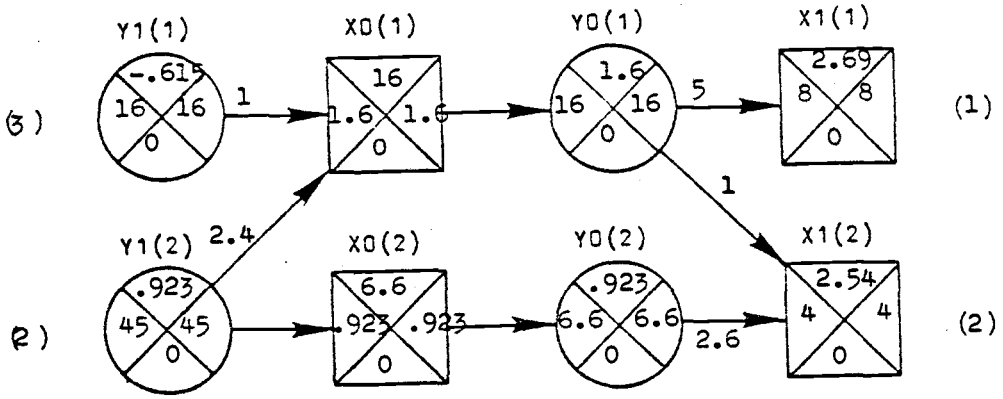


Figure 6-7. The Second Iteration for Illustrative Example 6.2.

X1 (2) = 6.6/2.6 = 2.54, X (2) = 2.54

X1 (1) = (16 - 2.54)/ 5 = 2.69, X (1) = 2.69

(b) Update resource variables.

Yo (1) = 8/5 = 1.6

Yo (2) = (4 - 1.6)/ 2.6 = .923

Y1 (2) = .923, Y (2) = .923

Y1 (1) = 1.6 - 2.4 • .923 = -.615, Y (3) = -.615

2. Update unlabeled network.

(a) Update resource residues:

S (1) = 19 - 2.69 - 4 • 2.54 = 6.15

(b) Update process residues:

(All process nodes are labeled)

The current solution is not feasible.

Iteration 3. (Figure 6-7)

(C.3.) The third resource variable is negative. In this case type (c) labeling is applied.

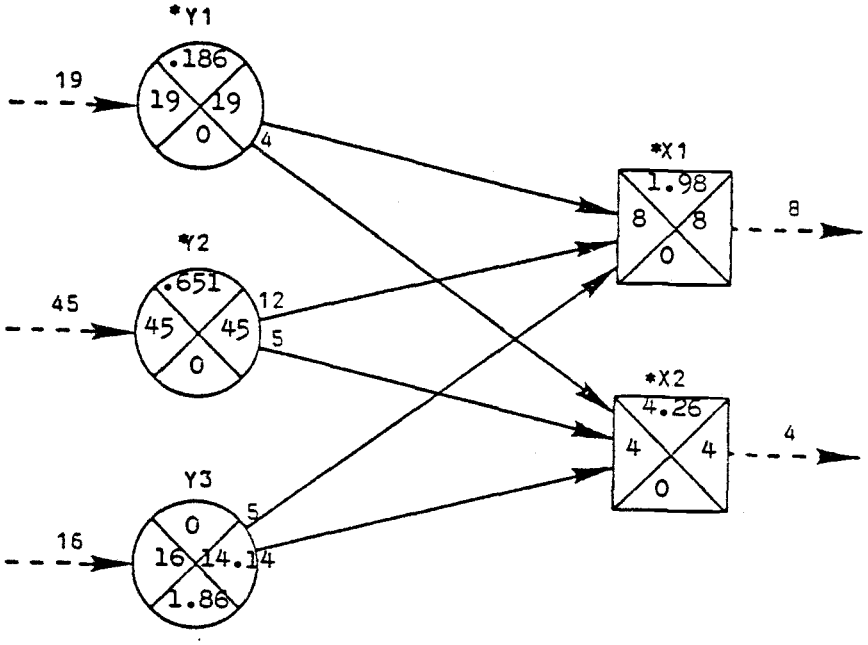
(L.3.) Let $i_1 = 3$. For this resource node:

$I_3 = (1, 2, 3)$ and $U_3 = ((1,5), (2,1))$

$I_3^* = (1,2)$, $I_3 = \emptyset$

(i) The perturbated values of the process variables are found from:

12 $X\varepsilon$ (1) + 5 $X\varepsilon$ (2) = 0



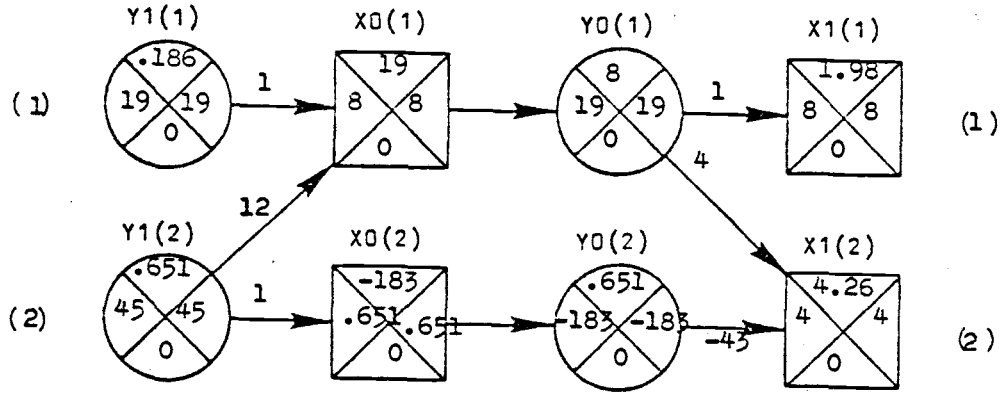


Figure 6-8. The Final RPM Network for Illustrative Example 6.2.

and $5\,X\varepsilon\,(1) + X\varepsilon\,(2) = -\varepsilon$

We have, $X\varepsilon\,(1) = -.384\varepsilon$ and $X\varepsilon\,(2) = .92\varepsilon$

Next, we compute t_1 as:

$t_1 = \min\,(2.69/\ .384) = 7$, and $j_1 = 1$

(ii) The updated residues are:

$S\varepsilon\,(1) = .384\varepsilon - 4 \cdot .92\varepsilon = -3.296\varepsilon$

The value of t_2 is computed next:

$t_2 = \min\,(6.15/\ 3.296) = 1.86$, and $i_2 = 1$

(iii) The value of t is:

$t = \min\,(7, 1.86) = 1.86$, i.e. $t = t_1$

In this case the resource node 1 becomes labeled and resource node 3 becomes unlabeled.

(F.3.) The dimension is $\rho = 2$. The labeled network is:

$l_{11} = 1$, $l_{21} = 12$, $l_{22} = 1$ and $u_{11} = 1$, $u_{12} = 4$ and $u_{22} = 5 - 12 \cdot 4 = -43$

(B.3.) The updated Bo and Co are: (Bo (1) = 19, Bo (2) = 45 (corresponding to the resource nodes), and Co (1) = 8, Co (2) = 4 (corresponding to the process nodes).

1. Update labeled network.

(a) Update process variables:

Xo (1) = 19

Xo (2) = 45 - 12 · 19 = -183

RPM test file (3X2)

Iteration No. 1 Objective Function 25.600

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	19.000	15.800	2	1.0 (1)	4.0 (2)		
2	<RES002		0.000	45.000	6.600	2	12.0 (1)	5.0 (2)		
3	<RES003	*	1.600	16.000	0.000	2	5.0 (1)	1.0 (2)		

Process Graph

1	PR001	*	3.200	8.000	0.000	3	1.0 (1)	12.0 (2)	5.0 (3)	
2	PR002		0.000	4.000	-2.400	3	4.0 (1)	5.0 (2)	1.0 (3)	

Iteration No. 2 Objective Function 31.692

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	19.000	6.153	2	1.0 (1)	4.0 (2)		
2	<RES002	*	0.923	45.000	0.000	2	12.0 (1)	5.0 (2)		
3	<RES003	*	-0.615	16.000	0.000	2	5.0 (1)	1.0 (2)		

Process Graph

1	PR001	*	2.692	8.000	0.000	3	1.0 (1)	12.0 (2)	5.0 (3)	
2	PR002	*	2.538	4.000	0.000	3	4.0 (1)	5.0 (2)	1.0 (3)	

Iteration No. 3 Objective Function 32.837

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	0.186	19.000	0.000	2	1.0 (1)	4.0 (2)		
2	<RES002	*	0.651	45.000	0.000	2	12.0 (1)	5.0 (2)		
3	<RES003		0.000	16.000	1.860	2	5.0 (1)	1.0 (2)		

Process Graph

1	PR001	*	1.976	8.000	0.000	3	1.0 (1)	12.0 (2)	5.0 (3)	
2	PR002	*	4.255	4.000	0.000	3	4.0 (1)	5.0 (2)	1.0 (3)	

Figure 6-9. Different Tableau for Illustrative Example 6.2.

X1 (2) = 183/43 = 4.26, X (2) = 4.26

X1 (1) = 19 - 4 · 4.26 = 1.98, X (1) = 1.98

(b) Update labeled resource variables.

Yo (1) = 8

Yo (2) = -(4 - 4 · 8)/ 43 = .651

Y1 (2) = .651, Y (2) = .651

Y1 (1) = 8 - 12 · .651 = .186, Y (1) = .186

2. Update unlabeled network.

S (3) = 16 - S · 1.98 - 4.26 = 1.86

The current solution is optimum. The value of the objective function is,

z = 8 · 1.98 + 4 · 4.26 = 32.8

VII. COMPUTATIONAL RESULTS

The purpose of this chapter is to evaluate the practicality of the new method, and provide statistical results based on test problems.

First, the "worst case" analysis based on all possible basic solutions was made. The three basic steps of the algorithm i.e. labeling, factoring and balancing were analyzed separately, in order to find an upper bound of their computational requirements, and find ways of avoiding nonnecessary operations. The criterion used to measure the amount of work involved was the number of multiplications and divisions at each iteration. Additions and subtractions were omitted, since they are executed much faster, they are proportional to the number of the other two arithmetic operations (multiplications and divisions), and mainly because they increase the complexity of the analysis. Typical computational speed for Wang 2200 on which the program was written are shown in Table 7.1.

For a number of sample problems, the number of iterations and total arithmetic operations (multiplications and divisions) were recorded and compared against simplex methods, using the MPOS (Multipurpose Optimization System) software package.

The Worst Case Analysis

The "worst case" analysis is often used as a theoretical measure of the efficiency of an algorithm. For extreme point algorithms, it is based on the total number of iterations, over all the

TABLE VII.1. EXECUTION TIMES FOR WANG 2200 B MODEL.*

CPU (Central Processing Unit) – System 2200, Model A or B

Variable Formats

Scalar Numeric Variable.
Numeric 1- and 2-dimension Array Variables.
Alphanumeric String Variable.
Alphanumeric 1- and 2-dimensional String Arrays.

Average Execution Times (Milliseconds)

Add/Subtract	0.8
Multiply/Divide	3.87/7.4
Square Root/e^x	46.4/25.3
$Log_e x/X^y$	23.2/45.4
Integer/Absolute Value	0.24/0.02
Sign/Sine	0.25/38.3
Cosine/Tangent	38.9/78.5
Arctangent	72.5
Read/Write Cycle	1.6μ sec

(Average execution times were determined using random number arguments with 13 digits of precision. Average execution times will be faster in most calculations with arguments having fewer significant digits.)

TAPE DRIVE –Model 2217

Stop/Start Time
0.09/0.05 sec
Capacity
522 bytes/ft (1712 bytes/m)
Recording Speed
7.5 IPS (19.05 cm/sec)
Search Speed
7.5 IPS (19.05 cm/sec)
Transfer Rate
326 characters/sec (approx.)
Inter-record Gap
0.6 in. (1.52 cm)
(Capacity and transfer rate include gaps and redundant recording.)

*WANG, Laboratories: Reference Manual, Model 2200 B.

extreme points and the number of operations per iteration. For the new algorithm the number of operations at each iteration depends not only on the size of the problem, and the density of the constraint matrix (100% density is assumed), but also on the size of the labeled network at each iteration. Figure 7.1 shows all possible combinations of the labeled network for problems with m resource, n process nodes, and ρ dimension of the labeled network: $0 \leq \rho \leq \min(m, n)$. For four resource nodes and four process nodes, $0 \leq \rho \leq 4$, and there are 70 combinations of basic solutions $(m + m)!/(m!\ m!)$ where $m = 4$. From all these basic solutions, one has no dimension, $\rho = 0$ (all nodes are unlabeled), 16 correspond to $\rho = 1$ (only one resource and one process node is labeled), and 36 correspond to $\rho = 2$, etc.

The next section discusses the number of arithmetic operations required per iteration for values of ρ ranging from 0 to min (m, n).

The Number of Operations per Iteration

In this section, an upper bound of the number of arithmetic operations (multiplications and divisions) is given for each of the three basic steps of the algorithm, i.e. labeling, factoring, and balancing.

For a given network with m resource and n process nodes, ρ is the number of labeled resource nodes (and also the number of labeled process nodes).

To find the total number of operations per iteration the following assumptions were made:

Labeled Networks

Density= 100 % I=m J=m

r/m	0	1	2	3	4	5	6	7	8	9	10	Total
(1)	1	1										2
(2)	1	4	1									6
(3)	1	9	9	1								20
(4)	1	16	36	16	1							70
(5)	1	25	100	100	25	1						252
(6)	1	36	225	400	225	36	1					924
(7)	1	49	441	1225	1225	441	49	1				3432
(8)	1	64	784	3136	4900	3136	784	64	1			12870
(9)	1	81	1296	7056	15876	15876	7056	1296	81	1		48620
(10)	1	100	2025	14400	44100	63504	44100	14400	2025	100	1	184756

Figure 7-1. All Possible Basic Solutions for an m x m Model.

(i) The constrained matrix is 100% dense.

(ii) Although type (a) labeling process is assumed, the number of arithmetic operations is the same in type (b) or type (c). The assumptions are:

(a) The ratio of the residue of each unlabeled resource node and its perturbated value must be found for all unlabeled resource nodes.

(b) A perturbated value is required for all the labeled process nodes and the ratio of the value of each basic process variable and its perturbated value must be calculated for all labeled process nodes.

(iii) At each iteration a complete factorization of the labeled network is required.

(iv) A complete network updating scheme is necessary, i.e. all the resource and process nodes are updated.

Under the above assumptions an upper bound of the operations required at each iteration is given analytically.

1. Labeling. For the process of labeling, the operation count is the sum of:

(i) ρ^2 operations for an explicit evaluation of the perturbated values of the labeled process variables, and ρ operations for finding the minimum ratio.

(ii) $\rho\ (m-\rho)$ operations for updating and $m-\rho$ operations for finding the minimum ratio of the residues of each unlabeled

resource node and its perturbated value.

Thus, the total amount of computations per labeling is:

$$\rho^2 + \rho + \rho(m-\rho) + m-\rho, \text{ or } \rho m + m$$

2. Factoring. For a complete factorization of a $\rho x \rho$ labeled network, the operation count is given by: $\rho(\frac{\rho^2-1}{3})$ (Isaacson, Keller, 1966, p.35).

3. Balancing. The amount of operations needed for balancing is the sum of:

(i) ρ^2 operations for an explicit solution of the labeled process variables. Also, ρ^2 operations for the labeled resource variables. Thus, a total of $2\rho^2$ operations.

(ii) ρ operations are required for updating the residue of an unlabeled resource node, i.e. a total of $\rho(m-\rho)$ for all unlabeled resource residues. Also, a total of $\rho(n-\rho)$ is required for updating the process residues.

Thus, each balancing process has an upper bound of $\rho(m + n)$ operations.

Statistical Data

A number of sample problems were selected and solved by the new algorithm. The sample problems were:

(a) Textbook problems: Sample problems were selected randomly from textbooks.

(b) Concocted problems: Sample problems were constructed to test the algorithm for:

(i) Unbounded solutions

(ii) Nonfeasible solutions and

(iii) Cycling on loops. Networks including chains of arcs which formed loops were tested by the new algorithm.

The statistical results for a set of sample problems are shown in Table 7.2. The number of iterations for each problem required by the new method and the simplex type methods(regular method and primal dual algorithm) were recorded.

The total amount of operations required for the solution (or for an indication of unboundedness or nonfeasibility conditions), are shown in Table 7.3.

TABLE VII.2. NUMBER OF ITERATIONS REQUIRED PER EXAMPLE PROBLEM.

Problem	m x n	Density %	Simplex	Primal - Dual	RPM
1	3 x 4	100	5	2	2
2	3 x 4	100	2	2	3
3	3 x 4	100	1	2	1
4	3 x 2	100	3	3	3
5	3 x 4	100	2	2	4
6	4 x 2	100	2	2	2
7	5 x 2	75	4	2	2
8	4 x 3	80	4	2	2
9	3 x 3	100	2	1	1
10	2 x 2	100	2	2	2
11	6 x 6	47	5	5	5
12	3 x 3	100	3	3	3
13	6 x 6	33	6	6	5
14	11 x 14	12	11	11	11
15	12 x 11	19	8	8	8
16	8 x 10	25	6	6	6

TABLE VII. 3. NUMBER OF OPERATIONS REQUIRED PER SAMPLE PROBLEM.

Problem	Size m x n	Density	Simplex	Primal - Dual	RPM
1	3 x 4	100	119	42	24
2	3 x 4	100	46	46	42
3	3 x 4	100	21	21	15
4	3 x 2	100	44	44	44
5	3 x 4	100	46	46	69
6	4 x 2	100	40	40	32
7	5 x 2	75	96	34	31
8	4 x 3	80	106	38	36
9	3 x 3	100	37	16	30
10	2 x 2	100	22	22	22
11	6 x 6	47	186	137	108
12	3 x 3	100	46	46	44
13	6 x 6	33	135	85	58
14	11 x 14	12	134	156	132
15	12 x 11	19	276	243	196
16	8 x 10	25	102	78	75

VIII. CONCLUSION

The purpose of this chapter is to discuss and evaluate the different criteria considered for the development of the new methodology, and to evaluate the experimental results based on sample problems.

Different ways of accelerating the optimization procedure by reducing the number of iterations and amount of arithmetic operations, and recommendations for future studies are also included in this chapter.

Feasibility of the New Algorithm

Let us first summarize the findings of our feasibility analysis of the new algorithm.

The labeling process described in Chapter Four alters the sets of basic resource and process nodes in such a way that the updated labeled network contains no redundant constraints (primal or dual), i.e. the labeled nodes form a non-singular array with rank ρ, where $\rho x \rho$ is the dimension of the basis. The factorization scheme described in Chapter Five guarantees a triangular structure for the labeled network since the matrix which corresponds to the labeled network is not singular.

The transformation of the labeled network into a triangular structure also guarantees that the primal and the dual basic solution can be found by backsubstitution. This is then used to update the nonbasic nodes of the network.

At each iteration the algorithm works towards optimality or towards feasibility. If the final solution is optimal but not feasible and no type (b) or type (c) labeling is possible, then the solution of the problem is infeasible. If the final solution is feasible but not optimal and no type (a) or type (c) labeling is possible, then the solution of the problem is unbounded. If the final solution is neither optimum nor feasible and no labeling is possible, then the solution of the problem is unbounded. If the final solution is neither optimum nor feasible and no labeling is possible then the problem is both unbounded and inconsistent.

Evaluation of the Criteria

The criteria discussed in Chapter One are:

1. Memory Space Requirements. Unlike most of the simplex type algorithms, the dimension of the basis for the new algorithm is not the same from one iteration to the next. The starting basis has zero dimension, and at each iteration the number of labeled resource nodes, ρ, defines the new dimensionality of the basis. The minimum number of the resource nodes m, and the process nodes n on the network, is an upper bound for ρ, i.e. $0 \leq \rho \leq r = \min(m, n)$.

In general, for problems with a dense constraint matrix (in its canonical form), the number ρ, on the average, is considerably less than r, when the ratio m/n is close to one. If the ratio m/n is very large or very small, then it is expected that ρ will get closer to r.

In Chapter Five, the decomposition method used for transforming the basic network into a triangular structure (LU decomposition) was

described in detail. Both arrays (L and U) can be stored on the same $\rho x \rho$ array in order to reduce the memory space requirements.

Under the new scheme the memory space requirements, on the average, are considerably less than that of a full basis.
For example, at each iteration the requirements (approximately) are:

(i) ρ^2 elements for storing the basis

(ii) $2 \cdot m$ elements for the resource variables and residues

(iii) $2 \cdot n$ elements for the process variables and residues

The resource and process graph of the network are saved on the same or different disks in a compact way.

2. Number of iterations. The number of iterations required for the new method is finite and depends on the size and the complexity of the problem under solution, as well as the hierarchy in which different types of labeling are applied.

When compared with simplex methods, the new algorithm, on the average, did not require more iterations than simplex type algorithms for sample problems discussed in Chapter Seven.

3. Arithmetic Operations per Iteration. At each iteration, the number of arithmetic operations varies depending upon:

(i) Size of the labeled network.

(ii) Density of the constraint matrix.

(iii) Status of the residues of the resource and process nodes.

(iv) Type of factorization required and the positions of the exchanged nodes.

The upper bound of operations required at each iteration is given in Chapter Seven ("worst case"). In practice, statistical data based on sample problems indicate that on the average the number of arithmetic operations is less than simplex type algorithms.

4. Round-off Errors. Many of the simplex type algorithms change the model representation at each iteration and errors occuring at every stage of the process directly affect the solution. If either the optimal feasible solution or its dual is degenerate or nearly so, it is possible that errors will affect the choice of the basic variables so that in the final tableau the segregation of variables into basic and nonbasic is incorrect. When the revised simplex is used the round-off errors are reduced. The difference in round-off errors arise from the way in which the original representation of the model is changed. In the revised simplex round-off errors occur only in the basic vectors, and the method is thus susceptible to numerical instability in the final stages of the solution (Jacobs, 1977).

In recent years it has become recognized that linear programming inversion routines for a full basis can be improved in accuracy when the Gaussian or Elimination Form of Inverse, rather than the Gauss-Jordan or Product form of Inverse is used (Forrest and Tomlin, 1971).

Under the new scheme, the round-off errors can be minimized by using the triangular factorization method for the reduced basis, and the original model representation. The factorization method can be modified in a way that:

(i) The maximum pivot element is used instead of the first nonzero in each row (column).

(ii) Double precision arithmetic can be used for the vectors Co (Bo), thus obtaining greater accuracy.

Basically, the round off errors occurring in the solution are due to operations made during the last iterations.

5. Limited Computer Requirements. The new method is designed in such a way that the original data can be stored as resource and process graph on one or more disks when a disk storage system is available. Depending on the size of the problem, the resource or process variables and residues can be stored in core or on the disk. The computer program may be further modified, so that at each iteration, the available memory space can be used to process a decomposed labeled network a segment at a time.

6. Network Structure. The new method, based on the RPM network technique eliminates the need for explicit logical variables (slack, surplus or artificial). When compaired with simplex algorithms the total number of explicit variables is considerably less.

Accelerating Methods

The new method described in Chapters Four through Eight is based on a complete network updating scheme and a single labeling at each iteration. It is also assumed that there was no initial basic

solution available, so that the zero solution is introduced at the first iteration. For large problems however, the method can be modified in a way that the number of iterations and arithmetic operations can be reduced. Some of the changes of the algorithm which may accelerate the optimization procedure are listed below:

(i) The algorithm can start with the primal problem working towards optimality on the relaxed network which contains only the less than or equal to constraints. After the optimum solution has been found for the relaxed network, the greater than constraints can be introduced and the algorithm continues working on both problems.

(ii) The labeling process may be changed in such a way that, at each iteration only the perturbated values for the process (resource) variables are computed and the feasibility (optimality) conditions are checked only for the candidate process(resource) node cluster. This approach has computational advantages, especially for networks which contain no loops.

(iii) Any number of unlabeled process (resource) nodes can be considered as candidates for labeling in a sequential order, as long as their clusters do not contain labeled nodes. This way more than one process (resource) node can become basic at each iteration.

(iv) Any number of pairs of nodes (one resource and one process node per pair), which form a path on the network may become labeled. Any basic path which does not contain loops has a triangular structure, so that the factorization process is trivial.

Recommendations for Future Research

In this research a network approach for the general linear programming problem has been proven feasible and the first practical experience with the new method has given satisfactory results. However, there is little doubt that significant improvements can be made to further improve the new methodology. The algorithm described has extreme point (labeled path) solutions. Theoretically the new method belongs to the "exponential" methods, since no attempt was made to bound the number of all possible combinations of labeled networks by a polynomial function.

For all practical purposes some promising areas for future research are listed below.

1. The implementation of the factorization. In general, the way in which a factorization method is implemented and updated depends critically upon the size of the problem it is designed to handle. In Chapter Five no attempt was made to include the most modern techniques available for very large and sparse problems. Sparse matrix preprocessors (i.e. rearrangement of columns and rows of a sparse matrix before factorization) can be applied to the labeled network to reduce the number of operations and storage requirements (Hellerman and Rarick, 1971).

2. Analysis of network paths. An analytic method that identifies the basic paths on the network may lead to near optimum starting basic solution.

3. Approximation method. For large sparse problems an approx-

imated solution based on approximation methods for solving linear systems of equations may lead to satisfactory results. A similar to overrelaxation method for solving linear systems of equations can be applied on both primal and dual problems. At each iteration the labeling process can be modified to include a set of basic process nodes with negative residues in such a way that the resulting labeled network has a triangular structure. The initial resource constants can be added to the resource residues, and the dual variables corresponding to labeled resource nodes, can be used to identify a new set of basic process nodes. In case of convergence the set of critical constraints and basic processes will give the solution of the problem.

4. Cycling. The new algorithm operates on basic solutions of the primal and dual models, by changing the sets of basic (labeled) nodes of the labeled network at each iteration. Since there is a finite number of basic solutions, the algorithm will terminate as long as no basis (i.e. the sets of labeled nodes which form the basis) will be repeated. A further study should be made to further research the cycling aspects.

5. Computational efficiency. The computational results given in Chapter VII were based on small sample problems. In general, the computational efficiency of the new algorithm cannot be decided unless a comparison is made with other methods, based on large linear problems, using the same computer for a true test of computational efficiency.

BIBLIOGRAPHY

Akyuz, F. A. and S. Utku. (1968) "An automatic node-relabeling scheme for bandwidth minimization of stiffness matrices." AIAA J. 6, 728-730.

Balas, E. (1966) "The Dual Method for the Generalized Transportation Problem." Management Science 12, 555-568.

Balas, E. and P. L. Hammer. (1964) "On the Generalized Transportation Problem." Management Science 11, 188-202.

Balinski, M. L. and R. E. Gomory. (1963) "A Mutual Primal Dual Simplex Method" in Recent Advances in Mathematical Programming. R. L. Graves and P. Wolfe, eds. McGraw-Hill, New York.

Barnes, J. W. and Crisp, R. M. (1977) "Linear Programming: A Survey of General Purpose Algorithms." AIIE, Vol. 7, no. 3, pp. 212-221.

Barr, R. S., D. Klingman and W. Raike. (1968) "Computational Simplifications through Topological Structure in Network and Distributions Models." Applied Mathematics for Management, No. AMM-13. Graduate School of Business, University of Texas at Austin.

Bartels, R. H. (1968) "A Numerical Investigation of the Simplex Method." Computer Science Dept., Stanford University. Technical Report No. CS-104 (July 31).

Bartels, R. H. (1971) "A stabilization of the simplex method." Num Math. 16, 414-434.

Bartels, R. H. and G. H. Golub. (1969) "The Simplex Method of Linear Programming using LU Decomposition." Comm. ACM 12, 266-268, 275-278.

Bartels, R. H., G. H. Golub and M. A. Saunders. (1970) "Numerical techniques in mathematical programming" in "Nonlinear Programming." J. B. Rosen, O. L. Mangasarian and K. Ritter, eds. Academic Press, London and New York, 123-176.

Beale, E. M. L. (1955) "Cycling in the Dual Simplex Algorithm." Naval Research Logistics Quarterly, Vol. 2, no. 4. December, 269-275.

Beale, E. M. L. (1963) "The Simplex Method using Pseudo-Basic Variables for Structural Linear Programs: in R. L. Graves and P. Wolfe, eds. Recent Advances in Math Progr. McGraw-Hill, New York.

Beale, E. M. L. (1970) "Sparseness in Linear Programming." Paper presented to the Oxford Symposium on Sparse Systems of Linear Equations. April.

Bellman, R. I. (1958) "On a Routing Problem." Quart. Appl. Math. 16, 87-90.

Bellmore, M., H. J. Greensberg and J. J. Jarvis. (1970) "Generalized Penalty Function Concepts in Mathematical Optimization." Oper. Res. 18, 229-252.

Bennett, J. M. (1966) "An Approach to Some Structured L. P. Problems." Oper. Res. 14(4), 636-645.

Bennett, J. M. and D. R. Green. (1966) "Updating the Inverse or the Triangular Factors of a Modified Matrix." Basser Computing Dept., University of Sydney. Technical Report No. 42, April.

Bennett, J. M. and D. R. Green. (1969) "An approach to some structured Linear Programming Problems." Oper. Res. 17(4), 749-750, August.

Ben-Israel, A. (1962) "Generalized Inverses and the Bott-Duffin Network Analysis." ONR Research Memo No. 66. Northwestern University, Ill.

Ben-Israel, A. (1960) "On Explicit Solutions of Interval Linear Programs." SIAM J. Math. 8, 12-22.

Ben-Israel, A. and A. Charnes. (1961) "Projection properties and Neumann-Euler expansions for the Moore-Peurose inverse of an arbitrary matrix. ONR Research Memo No. 40. Northwestern University, Ill.

Ben-Israel, A. and A. Charnes. (1968) "An Explicit Solution of a Special Class of Linear Programming Problems." Oper. Res. 1166-1175.

Ben-Israel, A. and P. D. Robers. (1968) "A Decomposition Method for Interval Linear Programming." Management Science 16(5), 374-387.

Bitran, G. R. and A. G. Novaes. (1973) "Linear Programming with a Fractional Objective Function." Oper. Res. 21(1), 22-29, February.

Björck, Å. (1976) "Methods for sparse linear least squares problems" in J. R. Bunch and D. J. Rose (1976), 177-199.

Boullion, Thomas L. and Patrick L. Odell. (1976) An Introduction to the Theory of Generalized Matrix Invertability. Texas Center for Research, Austin, Texas.

Brayton, R. K., F. G. Gustavson and R. A. Willoughby. (1969) "Some Results on Sparse Matrices." Report RC-2332. IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., February 14.

Brølund, O. E. and T. L. Johnsen. (1974) "QR factorization of partitioned matrices." Comput. Meth. Appl. Mech. Eng. 3, 153-172.

Buchet, J. (1970) "How to Take into Account the Low Density of Matrices to Design a Mathematical Programming Package-Relevant Effects on Optimization and Inversion Algorithms." Paper presented to the Oxford Symposium on Sparse Systems of Linear Equations, April.

Bunch, J. R. (1974) "Partial pivoting strategies for symmetric matrices." SIAM J. Numer. Anal. 11, 521-528.

Bunch, J. R. and D. J. Rose (eds.) (1976) "Sparse matrix computations." Proc. of Conf. held at Argonne National Laboratory, September 9 - 11, 1975. Academic Press, New York.

Businger, P. A. (1970) "Updating a singular-value-decomposition." BIT 11, 376-385.

Businger, P. and G. H. Golub. (1965) "Linear least squares solutions by Householder transformations." Num. Math. 7, 269-276.

Cabot, A. Victor, Richard L. Frauds, and Michael A. Stary. (1970) "A Network Flow Solution to a Rectilinear Distance Facility Location Problem." AIIE Transactions 2(2), June, 132-141.

Camion, P. (1965) "Characterization of Totally Unimodular Matrices." Proceedings of American Mathematical Society, 16, 1068-1073.

Chaiho, Kim. (1969) "Papameterizing an Activity Vector in L.P." University of Santa Clara, California.

Chandy, K. M. and Tacken Lo. (1973) "The Capacitated Minimum Spanning Tree." Networks 3(2), 173-181.

Charnes, A. and A. Ben-Israel. (1963) "Contributions to the Theory of Generalized Inverses." J. Soc. Industr. Appl. Math. 11(3), 667-699.

Charnes, A. and W. W. Cooper. (1961) Management Models and Industrial Applications of Linear Programming, V.I and II. John Wiley & Sons, N.Y.

Charnes, A. and C. E. Lemke. (1954) "Computational Theory of Linear Programming: The Bounded Variables Problem." ONE Research Memo, No. 10. Graduate School of Industrial Administration, Carnegie-Mellon University.

Clasen, R. J. (1968) "The Numerical Solution of Network Problems Using the Out-of-Kilter Algorithm." RAND Corporation Memorandum RM-5456-PR, CA.

Cline, R. E. and L. D. Pyle. (1973) "The Generalized Inverse in L.P.--An Intersection Projection Method and the Solution of a Class of Structured Problems." SIAM J. Appl. Math 24(3), May.

Clint, M. and A. Jennings. (1970) "The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration." Comp. J. 13, 76-80.

Clive, R. E. (1963) Representations for the Generalized Inverse of Matrices with Applications in Linear Programming. Doctoral Thesis, Purdue University. Lafayette, Indiana.

Clive, R. E. (1964) "Representations for the Generalized Inverse of a Partitioned Matrix." Journal Soc. Industr. Appl. Math. 12, 588-600.

Commoner, F. G. (1973) "A Sufficient Condition for a Matrix to be Totally Unimodular." Networks, 3, 351-365.

Cullum, J. and W. E. Donath. (1974) "A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices." Proc. IEEE Conf. on Decision and Control. Phoenix, Arizona.

Curtis, A. R. and J. K. Reid. (1971a) "Fortran subroutines for the solution of sparse sets of linear equations." AERE Report R.6844. HMSO, London.

Curtis, A. R. and J. K. Reid. (1971b) "The solution of large sparse unsymmetric systems of linear equations." J. Inst. Math. Appl. 8, 344-353.

Curtis, A. R. and J. K. Reid. (1972) "On the automatic scaling of matrices for Gaussian elimination." J. Inst. Math. Appl. 10, 118-124.

Cuthill, E. and J. McKee. (1969) "Reducing the bandwidth of sparse symmetric matrices." Proc. 24th National Conf. ACM. Brandon Systems Press, N.J., 157-172.

Daniel, J. W., W. B. Gragg, L. Kaufman, and G. W. Stewart. (1976) "Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization." Math. Comput. 30, 772-795.

Dantzig, G. B. (1955) "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming." Econometrica, Vol. 23, no. 2. April, 174-183.

Dantzig, G. B. (1963) "Linear Programming and Extensions." Princeton University Press, Princeton, N.J.

Dantzig, G. B. (1977) "Linear Programming: Its Past and Its Future." An unpublished paper presented at the annual meeting of Science Perspectives.

Dantzig, G. B. and R. P. Harvey. (1969) "Sparse Matrix Techniques in Two Mathematical Programming Codes" in Sparse Matrix Proceedings. IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 85-99.

Dantzig, G. B. and W. Orchard-Hays. (1954) "The Product Form of the Inverse in the Simplex Method." Mathematical Tables and Aids to Computational. Vol. 8, 64-67.

Dantzig, G. B. and P. Wolfe. (1959) "Decomposition Principle for Linear Programs." The RAND Corp., California, November.

Dantzig, G. B. and P. Wolfe. (1960) "Decomposition Principle for Linear Programs." Oper. Res., 8, 101-111.

Dantzig, G. B. and R. M. Van Slyke. (1967) "Generalized Upper Bounding Techniques." Journal of Computer and System Sciences, 1, 213-226.

Dantzig, G. B., L. R. Ford, and D. F. Fulkerson. (1956) "A Primal-Dual Algorithm for Linear Programs, in Linear Equalities and Related Systems." Kuhn and Tucker, eds. Princeton University Press.

Dittman, J. P. (1973) "Surrogated Linear and Quadratic Programming." Ph.D. Dissertation, University of Missouri, Columbia.

Duff, I. S. and J. K. Reid. (1974) "A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination." J. Inst. Math. Appl. 14, 281-291.

Duff, I. S. and J. K. Reid. (1976) "A comparison of some methods for the solution of sparse overdetermined systems for linear equations." J. Inst. Math. Appl. 17, 267-280.

Duff, I. S. and J. K. Reid. (1977) "An implementation of Tarjan's algorithm for the block triangularization of a matrix." To appear in ACM Trans. Math. Software.

Duff, I. S., A. M. Erisman, and J. K. Reid. (1976) "On George's nested dissection method." SIAM J. Numer. Anal. 13, 686-695.

Eckstein, V. (1954) "The Input-Output System--Its Nature and Use." Economic Activity Analysis. O. Morgestern, Editor. John Wiley & Sons.

Edmonds, J. and R. M. Karp. (1972) "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." ACM 19(2), 248-264, April.

Elfving, T. (1975) "Some numerical results obtained with two gradient methods for solving the linear least square problems." Dept. Math. Linköping. Report LiH-MAT-R-75-5.

Elmaghraby, Salah E. (1970a) "The Theory of Networks and Management Science, Part I. Management Science 17(1), 1-34, September.

Elmaghraby, Salah E. (1970b) "The Theory of Networks and Management Science, Part II." Management Science 17(2), 354-371.

Erisman, A. M. and J. K. Reid. (1974) "Monitoring the stability of the triangular factorization of a sparse matrix." Numer. Math. 22, 183-186.

Everett, H. (1962) "Generalized Language Multiplier Method for Solving Problems of Optimum Allocation of Resources." Institute of Defense Analyses, August 20.

Fletcher, R. and M. P. Jackson. (1974) "Minimization of a quadratic function of many variables subject only to lower and upper bounds. J. Inst. Maths. Applics. 14, 159-174.

Fletcher, R. and M. J. D. Powell. (1975) "On the modification of LDL^T factorizations." Maths. of Comp. 29.

Florian, M. and M. Klein. (1970) "An Experimental Evaluation of Some Methods of Solving the Assignment Problem." Canadian Oper. Res. Soc. 8, 101-108.

Ford, L. and D. Fulkerson. (1957a) "Solving the Transportation Problems." Management Science 3, 24-32.

Ford, L. and D. Fulkerson. (1957b) "A Primal-Dual Algorithm for the Capacitated Hitchcock Problems." Naval Research Log. Quart. 4, 47-54.

Ford, L. and D. Fulkerson. (1962) "Flows in Networks." Princeton University Press, Princeton, N.J.

Forrest, J. J. and J. A. Tomlin. (1971) "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method." NATO Conference on Large-Scale Math. Programming. Elsinore, Denmark, July.

Forrest, J. J. and J. A. Tomlin. (1972) "Updating triangular factors of the basis to maintain sparsity in the product-form simplex method." Math. Prog. 2, 263-278.

Forsythe, G. and C. B. Moler. (1967) Computer Solution of Linear Algebraic Equations. Prentice-Hall, Englewood Cliffs, N.J.

Frorian, M. and P. Robers. (1971) "A Direct Search Method to Locate Negative Cycles in a Graph." Management Science 17(5), 307-310, January.

Fulkerson, D. (1961) "An Out-of-Kilter Method for Minimal Cost Flow Problems." J. SIAM 9, 18-27.

Fulkerson, D. R. and P. Wolfe. (1962) "An algorithm for scaling matrices." SIAM Rev. 4, 142-146.

Garfinkel, R. S. and G. L. Nemhauser. (1972) Integer Programming. John Wiley & Sons, New York.

Garfinkel, R. S. and P. L. Yu. (1974) "A Primal-Dual Surrogate Simplex Algorithm: Comparison with the Criss-Cross Method." CSS 73-12, May.

Gass, S. I. (1958) Linear Programming, Methods and Applications. McGraw-Hill Book Co., Inc., New York.

Gay, D. M. (1974a) "On Scolnik's Proposed Polynomial Time Linear Programming Algorithm." TR73-190. Computer Science Department, Cornell University, June.

Gay, D. M. (1974b) "More Remarks on Scolnik's Approach to Linear Programming." TR74-207. Computer Science Dept., Cornell University.

Geoffrion, A. M. (1970) "Primal-Resource-Directive Approaches for Optimizing Nonlinear Decomposable Systems." Oper. Res. 18(3), 375-403, May-June.

Geoffrion, A. M. (1971) "Duality in Nonlinear Programming. A Simplified Applications-Oriented Development." SIAM Review, 13, 1-37.

George, J. A. (1973) "Nested dissection of a regular finite element mesh." SIAM J. Numer. Anal. 10, 345-363.

George, J. A. and J. W. H. Liu. (1975) "Some results on fill for sparse matrices," SIAM J. Numer. Anal. 12, 452-455.

Gentleman, W. M. (1973) "Least squares computations by Givens transformations without square roots." J. Inst. Math. Appl. 12, 329-336.

Gentleman, W. M. (1975) "Row elimination for solving sparse linear systems and least squares problems" in "Numerical Analysis Dundee 1975." G. A. Watson (ed.). Lecture Notes in mathematics, 506, Springer-Verlag, 122-133.

Gibbs, N. E., W. G. Poole, Jr. and P. K. Stockmeyer. (1976) "An algorithm for reducing the bandwidth and profile of a sparse matrix." SIAM J. Numer. Anal. 13, 236-250.

Gill, P. E. (1974) "Recent developments in numerically stable methods for linear programming." Bull. Inst. Maths. Applics. 10, 180-186.

Gill, P. E. and W. Murray. (1973) "A numerically stable form of the simplex algorithm." J. Linear Algebra Applics. 7, 99-138.

Gill, P. E. and W. Murray. (1974a) "Newton-type methods for unconstrained and linearly constrained optimization." Mathematical Programming 7, 311-350.

Gill, P. E. and W. Murray. (1974b) "Newton-type methods linearly constrained optimization" in "Numerical Methods for Constrained Optimization." P. E. Gill and W. Murray (eds.). Academic Press, London and New York, 29-65.

Gill, P. E. and W. Murray. (1974c) "Quasi-Newton methods for linearly constrained optimization," in "Numerical Methods for Constrained Optimization." P. E. Gill and W. Murray (eds.). Academic Press, London and New York, 67-90.

Gill, P. E. and W. Murray. (1974d) "Methods for large-scale linearly constrained problems," in "Numerical Methods for Constrained Optimization." P. E. Gill and W. Murray (eds.). Academic Press, London and New York, 93-147.

Gill, P. E. and W. Murray. (1976a) "The orthogonal factorization of a large sparse matrix" in "Sparse Matrix Computations." J. Bunch and D. J. Rose (eds.). Academic Press, London and New York.

Gill, P. E. and W. Murray. (1976b) "Minimization of a nonlinear function subject to bounds on the variables." Nat. Phys. Lab. Report. NAC 72.

Gill, P. E. and W. Murray. (1977a) "The computation of Lagrange multiplier estimates." Paper presented at the symposium on "Matrix Methods in Optimization." Argonne National Laboratory, June, 1976. Nat. Phys. Lab. Report NAC 75.

Gill, P. E. and W. Murray. (1977b) "Numerically stable methods for quadratic programming." Nat. Phys. Lab. Report NAC 76.

Gill, P. E., W. Murray and M. A. Saunders. (1975) "Methods for computing and modifying the LDV factors of a matrix." Math. Comput. 29, 1051-1077.

Gill, P. E., G. H. Golub, W. Murray and M. A. Saunders. (1974) "Methods for modifying matrix factorizations." Math. Comput. 28, 505-535.

Glicksman, S., L. Johnson, and L. Eselson. (1970) "Coding the Transportation Problems." Naval Research Logistics Quarterly, 7.

Glover, F. (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem." Oper. Res. December, 879-919.

Glover, F. (1968) "Surrogate Constraints." Oper. Res. 16, 741-749.

Glover, F. and D. Karney. (1972) "The Augmented Predecessor Index Method for Locating Stepping-Stone Paths and Assigning Dual Prices in Distribution Problems." Transportation Science, Vol. 6, no. 2. May, 171-179.

Glover, F. and D. Karney. (1974) "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems." Management Science 29(5), January, 793-813.

Glover, F. and D. Klingman. (1965) "Higher Order Tree Dual Approximation Methods for the Distribution Problem." Management Science, 14, 562-578.

Glover, F. and D. Klingman. (1970a) "Double Pricing Dual and Feasible Start Algorithm for the Capacitated Transportation (Distribution) Problem." Applied Math. for Management, No. AMM-20, WP 70-58.

Glover, F. and D. Klingman. (1970b) "Locating Stepping-Stone Paths in Distribution Problems via the Predecessor Index Method." Transportation Sci. 4, 220-225.

Goldfarb, D. (1975) "On the Bartels-Golub decomposition for linear programming cases." AERE Harwell Report CSS 18.

Goldfarb, D. and J. K. Reid. (1975) "A practicable steepest-edge simplex algorithm." AERE Harwell Report CSS 19.

Graves, R. L. (1966) "A Principal Pivoting Simplex Algorithm for Linear and Quadratic Programming." University of Chicago, Illinois, October.

Graves, G. W. and R. D. McBride. (1976) "The Factorization Approach to Large-Scale Linear Programming." Math. Progr. 10, 91-110.

Greenberg, J. H. (1969) "On the Relations of Everett's Theorem and Conjugate Functions." Tech. Memo CP-69002, SMU, Dallas, Texas.

Greenberg, H. and W. Pierskalla. (1969) "Surrogate Mathematical Programming." Southern Methodist University, Dallas, Texas.

Greville, T. N. E. and A. Ben-Israel. (1974) Generalized Inverses: Theory and Applications. Wiley-Interscience.

Grigoriadis, M. D. (1971) "A Dual Generalized Upper Bounding Technique." Management Science 17(5), 269-285.

Hachtel, G. D. (1972) "Vector and matrix variability type in sparse matrix algorithms" in D. J. Rose and R. A. Willoughby, 53-64.

Hall, M. (1957) "An algorithm for distinct representatives." Amer. Math. Monthly 63, 716-717.

Hammarling, S. (1974) "A note on modifications to the Givens plane rotation." J. Inst. Maths. Applics. 13, 215-218.

Hamming, R. W. (1971) "Introduction to Applied Numerical Analysis." McGraw-Hill, New York.

Harris, M. Y. (1970) "A Mutual Primal-Dual Linear Programming Algorithm." Naval Research Logistics Quarterly 19(2), June, 199-206.

Harris, P. M. J. (1973) "Pivot selection methods of the Devex LP code." Math. Prog. 5, 1-28.

Hadley, G. (1974) Linear Programming. Addison-Wesley Publishing Co., Inc., Reading, Massachusetts.

Hellerman, E. and D. Rarick. (1971) "Reinversion with the Pre-assigned Pivot Procedure." Math. Progr. 1, 195-216.

Inoue, M. S. (1974) "Visual Identification of Kuhn-Tucker Conditions on RPM Networks." presented at the Second Systems Engineering Conference Minneapolis, Minnesota.

Inoue, M. S. (1975) "Application of RPM Methodology to a Multiobjective Management of Productivity," Proc. of Annual Systems Engineering Conference, Las Vegas, Nevada, November.

Inoue, M. S. and J. L. Riggs (1972) "Resource Planning and Management Network," Proceedings of the International Symposium on Systems Engin. and Analysis, Purdue Univ., Lafayette, Indiana.

Inoue, M. S. and J. L. Riggs (1975) "Graphical Approach to Resource Allocations in Production Systems," Proc. of the Third International Conference on Production Systems, Amherst, Ma., August.

Isaacson, E. and H. B. Keller (1966) "Analysis of Numerical Methods." John Wiley & Sons, New York, 578 p.

Jacobs, D. (1977) "The State of the Art in Numerical Analysis." Academic Press, New York, 978 p.

Jennings, A. and W. J. Stewart. (1975) "Simultaneous iteration for partial eigensolution of real matrices." J. Inst. Math. Appl. 15, 351-361.

Khachian, L. G. (1979) "A Polynomial Algorithm in Linear Programming." DOKLADY Akademia Nauk, USSR 224, no. 5, pp1093-96 (translated as Soviet Mathematics DOKLADY 20, 191-194.

Lemke, C. E. (1954) "The Dual Method of Solving Linear Programming Problems." Naval Research Log. Quart. 1, 36-47.

McBride, R. D. (1975) "Linear Programming with Linked Lists and Automatic Guberization." Working Paper No. 8175. University of Southern California, July.

McBride, R. D. (1977) "A Bump Triangular Dynamic Factorization Algorithm for the Simplex Method." Working Paper No. 19. University of Southern California, November.

McBride, R. D. (1978) "A Spike Collective Dynamic Factorization Algorithm for the Simplex Method." Management Science 24(10), June.

McCormick, G. P. (1970) "A second-order method for the linearly constrained non-linear programming problem," in "Nonlinear Programming." J. B. Rosen, O. L. Mangasarian and K. Ritter (eds.). Academic Press, London and New York, 207-243.

Meijerink, J. A. and H. A. van der Vorst. (1976) "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix." Academic Computer Centre, Utrecht. Technical Report TR.1.

Mueller, R. K. and L. Cooper. (1965) "A Comparison of the Primal Simplex and Primal-Dual Algorithms for Linear Programming." Communications of the ACM 8(11), 682-686.

Paige, C. C. (1972) "Computational variants of the Lanczos method for the eigenproblem." J. Inst. Math. Appl. 10, 373-381.

Peters, G. and J. H. Wilkinson. (1969) "Eigenvalues of $Ax = \lambda Bx$ with band symmetric A and B." Comp. J. 12, 298-404.

Peters, G. and J. H. Wilkinson. (1970) "The least squares problem and pseudo-inverses." Comp. J. 13, 309-316.

Penrose, R. (1955) "A generalized inverse for matrices." Proc. Cambridge. Philosophical Soc., 51, 406-413.

Pyle, L. D. (1960) The Generalized Inverse in Linear Programming. Ph.D. Thesis. Purdue University, Lafayette, Indiana.

Pyle, L. D. (1964) "Generalized Inverse Computations Using the Gradient Projection Method." Journal of the ACM, 11(4), 422-428.

Pyle, L. D. (1967) "A Generalized Inverse e- Algorithm for Intersection Projection Matrices with Applications." Numerische Mathematik, 10, 86-102.

Pyle, L. D. (1972) "The Generalized Inverse in Linear Programming Basic Structure." SIAM Journal of Applied Mathematics 22, 335-355.

Pyle, L. D. and R. E. Cline. (1973) "The Generalized Inverse in Linear Programming--Interior Gradient Projection Methods." SIAM Journal of Applied Mathematics 24(4), 511-534.

Ralston, A. (1965) "A First Course in Numerical Analysis." McGraw-Hill, New York, 578 p.

Reid, J. K. (ed.). (1971a) "Large Sparse Sets of Linear Equations." Proc. of Conf. at Oxford, April 1970. Academic Press, London.

Reid, J. K. (1971b) "A note on the stability of Gaussian elimination." J. Inst. Math. Appl. 8, 374-375.

Reid, J. K. (1972) "Two FORTRAN subroutines for direct solution of linear equations whose matrix is sparse, symmetric and positive definite." AERE Report R.7119, HMSO, London.

Reid, J. K. (1975) "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases." Report CSS 20. Comp. Sci. and Systems Division, AERE. Oxfordshire, England.

Riggs, J. L. and M. S. Inoue (1975) Introduction to O. R. and M. S., McGraw-Hill, New York.

Robers, P. D. (1968) "Interval linear programming." Ph.D. Thesis. Department of the IE and MS. Northwestern University, August.

Robers, P. D. and A. Ben-Israel. (1968) "A Suboptimization method for interval linear programming: A new method for linear programming." Systems Research Memo, No. 206. Northwestern University, June.

Rose, D. J. and J. R. Bunch. (1972) "The role of partitioning in the numerical solution of sparse systems," in D. J. Rose and R. A. Willoughby (1972), 177-187.

Rose, D. J. and R. A. Willoughby (eds.). (1972) "Sparse Matrices and Their Applications." Proc. of Conf. at IBM Research Center, N.Y. September 9th-10th, 1971. Penum Press, New York.

Rosen, J. B. (1961) "The gradient projection method for non-linear programming, Part II--Non-linear constraints." J. Soc. Ind. Appl. Maths. 9, 514-532.

Rosen, R. (1968) "Matrix bandwidth minimization." Proc. 23rd National Conf. ACM. Brandon Systems Press, New Jersey, 585-595.

Rutenberg, D. P. (1972) "Generalized Networks, Generalized Upper Bounding and Decomposition of the Convex Simplex Method." Man. Sci. 16(5).

Rutishauser, H. (1969) "Computational aspects of F. L. Bauer's simultaneous iteration method." Numer. Math. 13, 4-13.

Rutishauser, H. (1970) "Simultaneous iteration methods for symmetric matrices." Numer. Math. 16, 205-223.

Saksena, C. P. and A. J. Cole. (1971) "The Bounding Hyperplane Method of Linear Programming." Opns. Res., February, 1-18.

Saunders, M. A. (1972a) "Lard-scale linear programming using the Cholesky factorization." Report STAN-CS-72-252. Computer Science Department, Stanford University, Stanford, California.

Saunders, M. A. (1972b) "Product form of the Cholesky factorization for large-scale linear programming." Report STAN-CS-72-301. Computer Science Department, Stanford University, Stanford, California.

Saunders, M. A. (1975) "The complexity of LU updating in the simplex method," in "The Complexity of Computational Problem-Solving." R. S. Anderson and R. P. Brent (eds.). Queensland University Press.

Saunders, M. A. (1976) "A fast, stable implementation of the simplex method using Bartels-Golub updating," in "Sparse Matrix Computations." J. Bunch and D. J. Rose (eds.). Academic Press.

Schwarz, H. R. (1968) "Tridiagonalization of a symmetric band matrix." Numer. Math. 12, 231-241.

Scoins, H. I. (1964) "The Compact Representation of a Routed Tree and the Transportation Problem." Symp. on Math. Programming, London.

Shor, N. Z. (1970) "Utilization of the Operation of Space Dilatation in Minimization of Convex Functions." Kibernetica 6, no. 1, Jan.-Feb., 6-12. Translated as Cybernetics 6, pp. 7-15.

Smith, D. K. (1969) "A Dynamic Component Suppression Algorithm for the Acceleration of Vector Sequences." Ph.D. Thesis. Purdue University, Lafayette, Indiana.

Speelpenning, B. (1973) "The generalized element method." Unpublished manuscript.

Spinivasan, V. (1972) "Accelerated Algorithm for Labeling and Relabeling of Trees with Application for Distr. Problems." Journal of the Association for Comp. Mach., Vol. 19, 4 Oct., 712-726.

Spinivasan, V. and G. L. Thompson. (1973) "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm Journal of the Association for Comp. Machinery.

Sposito, V. A. (1970) "Solutions of a Special Class of Linear Progr. Problems." Management Science 16, 386-388.

Staats, Glenn E. (1970) "Computational Aspects of Geometric Programming with Degrees of Difficulty." Ph.D. Dissertation, University of Texas, Austin.

Stewart, G. W. (1974) "Modifying pivot elements in Gaussian elimination." Math. Comp. 28, 537-542.

Stewart, G. W. (1976) "Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices." Num. Math. 25, 123-136.

Talacko, J. V. and R. T. Rockefeller (1960) "A Compact Simplex Algorithm and a Symmetric Algorithm for General Linear Programs." An unpublished paper, Marquette University.

Tarjan, R. (1972) "Depth first search and linear graph algorithms." SIAM J. Comput. 1, 146-160.

Tomlin, J. A. (1972) "Pivoting for size and sparsity in linear programming inversion routines." J. Inst. Math. Appl. 10, 289-295.

Wagner, H. M. (1956) "A Comparison of the Original and Revised Simplex Methods." MIT, Mass.

Wilkinson, J. H. (1961) "Error analysis of direct methods of matrix inversion." J. ACM 8, 281-330.

Wilkinson, J. H. (1965) "The Algebraic Eigenvalue Problem." Oxford University Press, London.

Wilkinson, J. H. (1967) "The Solution of Ill-Conditioned Linear Equations." Mathematical Methods for Digital Computers, Chap. 3, Vol. 2. John Wiley & Sons, Inc., New York.

Wolfe, C. S. (1973) "Linear Programming with FORTRAN", Scot, Foresman and Co., Illinois.

Wolfe, P. (1965) "The Co mposite Simplex Algorithm. "SIAM Review 7(1), 42-54.

Zangwill, W. I. (1967) "The Convex Simplex Method." Management Science, 14(3), November.

Zionts, S. (1969) "The Criss-Cross Method for Solving Linear Programming Problems." Management Science, 15(7), 426-445.

Zionts, S. (1972) "Some Empirical Tests of the Criss-Cross Method." Management Science, 19(4), 406-410.

APPENDIX A

RPM NOTATION

APPENDIX A

RPM NOTATION

RESOURCE PLANNING AND MANAGEMENT (RPM)

Relationship of RPM to the Traditional Linear Programming Model

Consider a linear programming model with n structural variables and m constraints.

$$\text{MAXIMIZE } Z_X = C \cdot X$$

$$\text{s.t. } A \cdot X \leq B$$

$$\text{where } X = \{x_j\}\ 1 \leq j \leq n \text{ and } x_j \geq 0$$

$$B = \{b_i\}\ 1 \leq i \leq m$$

$$C = \{c_j\}\ 1 \leq i \leq m$$

$$A = \{a_{ij}\}\ 1 \leq i \leq m;\ 1 \leq j \leq n$$

Let the A matrix, B and C vector separated into positive and negative elements so that:

$$A = A^+ - A^- \text{ where } A^+ = a_{ij}^+;\ a_{ij}^+ \geq 0$$

$$A^- = a_{ij}^-;\ a_{ij}^- \geq 0$$

$$a_{ij}^+ \cdot a_{ij}^- = 0 \text{ and } a_{ij}^+ - a_{ij}^- = a_{ij}$$

$$B = B^+ - B^- \text{ where } B^+ = b_i^+;\ b_i^+ \geq 0$$

$$B^- = b_i^-;\ b_i^- \geq 0$$

$$b_i^+ \cdot b_i^- = 0 \text{ and } b_i^+ - b_i^- = b_i$$

$$\text{and } C = C^+ - C^- \text{ where } C^+ = c_j^+;\ c_j^+ \geq 0$$

$$C^- = c_j^-;\ c_j^- \geq 0$$

$$c_j^+ \cdot c_j^- = 0 \text{ and } c_j^+ - c_j^- = c_j$$

The linear programming problem can then be written.

$$\text{MAXIMIZE } Z_X = C^+ X - C^- \cdot X$$

$$s.t.\ A^{+} \cdot X + B^{-} \leq A^{-} \cdot X + B^{+}$$

The RPM network representation of this linear programming problem will identify each inequality as a resource constraint, and each x_j as a process decision variable. A solid arrow joins node "i" to node "j" if $a_{ij}^{+} > 0$. The arrow points from "j" to "i", if $a_{ij}^{-} > 0$. A dashed arrow symbolizes an exogenous flow from a source terminal to node "i" if $b_i^{+} > 0$, and a reversed flow if $b_i^{-} > 0$. A dashed arrow symbolizes an endogenous flow from node "j" to the sink terminal if $c_j^{+} > 0$, and a reversed flow if $c_j^{-} > 0$.

Components of RPM

As can be seen in Figure A-1, the RPM network graphically portrays the linear relationships through the use of three basic components.

I. The first component of the RPM network is the resource which is represented by a circle and interpreted as a constraint of the linear programming model.

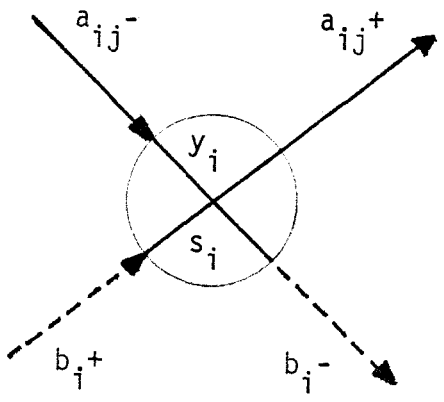


Figure A-1. Resource Node.

The constraints can be thought of in terms of the amount of

resources available ($\sum_{j=1}^{n} a_{ij}^{-} - x_j + b_i^{+}$) for future use ($\sum_{j=1}^{n} a_{ij}^{+} x_j + b_i^{-}$), or limit imposed upon subsequent processes x_j (where $a_{ij}^{+} > 0$). The resource node is divided into four sections. The top of the resource node (y_i) represents the "Shadow Price" or imputed value of the resource computed as the dual variable associated with the linear programming model, or Lagrangian Multiplier for that constraint. The bottom of the resource node S_i represents the "Residue" or the slack or surplus variable corresponding to the linear programming model. It may be interpreted as the unused portion of the resource or

$$S_i = \sum_{j=1}^{n} a_{ij}^{-} x_j + b_i^{+} - \sum_{j=1}^{n} a_{ij}^{+} x_j - b_i^{-}$$

The sections on the right and left side of the resource node can be used in order to insure than the input into the node ($\sum_{j=1}^{n} a_{ij}^{-} x_j + b_i^{+}$) is equal to the output ($\sum_{j=1}^{n} a_{ij}^{+} x_j + b_i^{-}$) from the node plus the amount of residue or slack, as indicated by S_i: i.e. total inflow = total outflow + S_i.

II. The second component of the RPM network is the process which is symbolized by a square node.

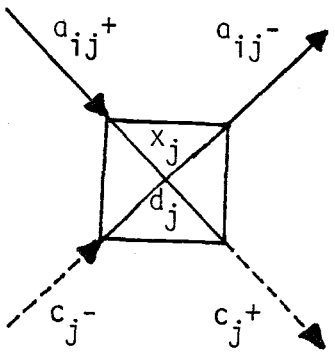


Figure A-2. Activity Node.

A process is interpreted as a decision activity which is actually representative of the action taken in order to achieve an end result. This node is also divided into four sections. The top of the square (x_j) designates the "Primal Level of Activity" which corresponds to the structural variable of the linear programming model: i.e. how many pounds of a product should be purchased, processed, frozen, etc. at the respective location in the model. The bottom of the square (d_j) represents the conventional ($z_j - c_j$) of the linear programming model, or the Lagrangian Multiplier associated with the non-negativity constraint of the primal variable. It can be interpreted as the "opportunity cost" or expected loss incurred by a non-basic activity if it were to be "forced" into solution by the management. The right and left sides of the activity node are available for manual verification that the input into the node ($\sum_{i=1}^{m} a_{ij}^{+} y_i + c_j^{-}$) is equal to the output from the node ($\sum_{i=1}^{m} a_{ij}^{-} y_i + c_j^{+}$) plus the residue, ($y_j$): i.e. total inflow outflow + y_j.

III. The third component of the RPM network is the objective function:

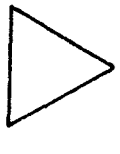


Figure A-3. Optimization Node.

which is represented by a triangle. Each maximization model will have one triangle as a source (on the left side of the network) indicating the minimizing objective for the corresponding dual model and one triangle as a sink (on the right side of the network) representing the maximizing primal objective function. This is symbolized by activity nodes connected to the maximization sink node (triangle with a square) through dashed arrows c_j:

$$\text{MAXIMIZE } z_x = \sum_{j=1}^{n} c_j^+ x_j - \sum c_j^- x_j$$

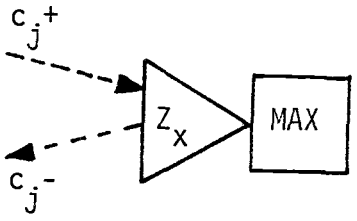


Figure A-4. Primal Maximization.

The word "MAX" on the square reveals that the primal objective function is to be maximized, and the value Z_x of the primal objective function is entered within the triangle.

The dual of the model under study then is to minimize the total value of input resources which are necessary to have the system operate under the optimal condition. This is represented by resource nodes connected to the minimization sink node (a triangle with a circle) through dashed arrows b_i: minimize

$$z_y = \sum_{i=1}^{m} b_i^+ y_i - \sum_{i=1}^{m} b_i^- y_i.$$

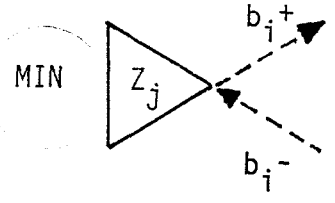


Figure A-5. Dual Minimization.

The word "min" in the circle designates the dual function to be minimized and the actual value of the objective function,

$$z_y = \Sigma\ b_i^-\ y_i - \Sigma\ b_i^-\ y_i,$$

is included within the triangle. In an RPM network, z_x always equals z_y.

The arrows depict the actual interrelationship of logic connections between the three previously described components (i.e. activity node, resource node, and maximizing/minimizing nodes). Two types of arrows are used in every RPM network.

The solid arrows represents the a_{ij} coefficient of the linear programming problem.

Figure A-6. Internal Flow Arrow.

When $a_{ij} = 0$, no arrow is shown. An arrow with $a_{ij} = 1$ may have its coefficient omitted from the network. The direction of the flow is depicted by the direction of the arrowhead.

The dashed arrow, as shown in Figure 7, represents an interaction between what is internal to the system under study (resources and activities) and what is outside the model itself (objective function).

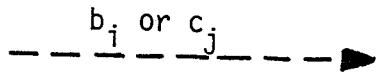


Figure A-7. Exogenous/Endogenous Flow Arrow.

Postulates of RPM

The necessary and sufficient conditions for the optimality and feasibility of a solution represented by an RPM network have been summarized as RPM postulates. The first two postulates are useful in the construction of the RPM networks.

The first postulate of RPM stipulates the conservation of resources. The total inflow at any activity or resource node must be larger than or equal to the total outflow from the same node. If the total inflow is larger than the total outflow there must be positive residue. If the total inflow into an activity or resource node is, however, exactly equal to the outflow, then the residue must be equal to zero.

$$\text{Total Input} \geq \text{Total Output}$$

$$\text{Total Input} = \text{Total Output} + \text{Residue}$$

$$\sum_j a_{ij}^{+} x_j + c_i^{-} = \sum_j a_{ij}^{-} x_j + c_i^{+} + d_j \text{ for } j = 1, 2, \ldots, n.$$

$$\sum_i a_{ij}^{-} y_i + b_i^{+} = \sum_i a_{ij}^{+} y_i + b_i^{-} + S_i \text{ for } i = 1, 2, \ldots, m.$$

The second postulate of RPM deals mainly with optimization of the system under study. The RPM network is to be optimized, either by maximizing the net effective output while maintaining the output at a given level.

$$\text{MAXIMIZE } Z_x = \Sigma c_j^{+} xj - \Sigma c_j^{-} x_j$$

$$\text{MINIMIZE } Z_y = \sum_i^j b_i^{-} y_i - \sum_i^j b_i^{+} y_i$$

A Resource Planning and Management (RPM) network can, therefore, be considered as a graphical representation of the interaction between activities and resources in order to achieve an optimal goal or end result. This end result can be considered as either a maximization of the primal, or minimization of the dual problem. Both the primal and the dual flows are represented simultaneously on the same RPM network.

Kuhn-Tucker Conditions

Let us consider a mathematical programming problem:

$$\text{MAXIMIZE} \quad Z_x(X)$$

$$\text{s.t.} \quad g_i(X) \leq b_i \qquad 1 \leq i \leq m_1$$

$$g_i(X) = b_i \qquad m_1+1 \leq i \leq m_2$$

$$g_i(X) \geq b_i \qquad m_2+1 \leq i \leq m$$

$$\text{where } X = x_j \qquad m+1 \leq j \leq m+n$$

and $x_j \geq 0$, where g_i are nonlinear functions.

Then a Lagrangian expression can be constructed to convert the constrained problem into an unconstrained equation.

$$\text{MAXIMIZE} \quad Z = Z_x - \sum_{i=1}^{m_1} y_i(g_i+s_i^2-b_i) + \sum_{i=m_1+1}^{m_2} y_i(b_i-g_i) + \sum_{i=m_2+1}^{m} y_i(g_i-b_i-s_i^2) + \sum_{j=m+1}^{m+n} y_j(x_j-s_j^2)$$

Note that the slack variables have been defined as s^2 to guarantee their nonegativity. To obtain the Kuhn-Tucker conditions, the Lagrangian expression is differentiated with respect to all independent variables and each is set equal to zero. (Inoue, 1974)

Differentiating with respect to the primal variables, x_j, we obtain:

$$\sum_{i=1}^{m_2} y_i \frac{\partial g_i}{\partial x_j} \geq \frac{\partial Z_x}{\partial x_j} + \sum_{i=m_2+1}^{m} y_i \frac{\partial g_i}{\partial x_j} \qquad \text{for } m+1 \leq j \leq m+n$$

Differentiating with respect to the Lagrangian multipliers (y_i or y_j), the original constraints can be recovered. Finally, differentiating Z with respect to the slack variables (s_i or s_j), the complementary slackness conditions are obtained.

$$0 = \frac{\partial Z}{\partial s_k} = -2s_k y_k \quad \text{where } 1 \leq k \leq m+n.$$

And since $x_j = s_j^2$ and we are at liberty to define $x_i = s_i^2$ for $1 \leq i \leq m$, the complemententary slackness can be expressed as:

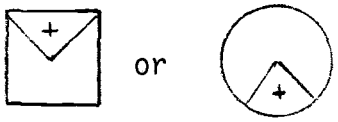
$x_k y_k = 0$ for $1 \leq k \leq m+n$. The sign of y's can also be determined[4].

When g_i for i from 1 to m are linear functions we may write:

$$\frac{\partial g_i}{\partial x_j} = a_{ij} \qquad \text{and} \qquad \frac{\partial Z_x}{\partial x_j} = c_j$$

The Kuhn-Tucker conditions for the linear programming problem and the corresponding RPM network presentation are given below:

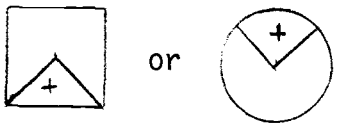
1. Feasibility condition. All primal variables must have nonnegative values.



(0 is permitted for +)

Figure A-8. Feasibility conditions.

2. Optimality condition. All dual variables must have nonnegative values.



(0 is permitted for +)

Figure A-9. Optimality conditions.

3. The complementary slackness property holds for all nodes.

---

4. All the Lagrange Multipliers must be nonnegative except the ones that correspond to the equal to constraints, which may have positive or negative values. The proof is in Inoue.

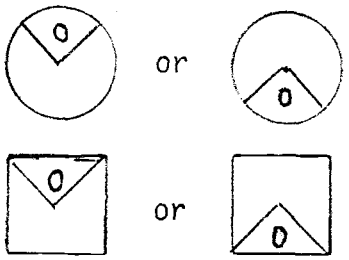


Figure A-10. Complementary slackness.

APPENDIX B

LISTINGS OF COMPUTER PROGRAMS

The programs included use WANG 2200 BASIC language (16k memory size) and they were designed to test the validity of the algorithm.

Some minor portions of the algorithm and Editor program are not included in the computer listings. A dual floppy disk system was used and the original data of the LP model are saved on disk.

Program Listing for RPM

```
 4010 COM BO$,M9,N9
 4040  SELECT  #1B10
     :<<: PRINT HEX(03)
 4100 REM=============================================REM
***RPM****(012980)GDS
 4120 M9=0
     :N9=0
 4130 DIM A8$(14)62,G8$(2)62,B$(100)8,C$(100)5
 4131 DIM D$(14)8,B(2),D(10),E(10),R(10),F(10),FO$8,RO$(8)
 4132 J9=99
 4150 SELECT <<: PRINT 005(80)
 4160 <<: PRINT HEX(03);
     :<<: PRINT "LINEAR PROGRAMMING (please enter data file";
     :<<: PRINT "in port F and program disk in port R  )";
     :<<: PRINT "(CONTINUE)to continue"
     :STOP
 4170 <<: PRINT
     :<<: PRINT "Please Enter Name Of Data File RUN ";BO$;
     :>>: INPUT BO$
     :AO$=BO$
     :LIMITS T#1,AO$,KO,MO,MO
     :GOSUB '41
 4180 REM=============================================REM

 4190 REM=============================================REM
*** store resource names on B$() , process names on C$()
 4200 DIM Q$45
     :Q$=STR(A8$(1),7,45)
     :INIT(20)A8$(1),A8$(2)
 4210 DIM F1$2
     :F1$=HEX(012C0120)
     :$UNPACK (D=F1$)A8$()TO B$()
     :@@: FOR I=1TO J9
     :B$(I)=B$(I+1)
     :<>: IF B$(I)=" " THEN  4220
     :↑↑: NEXT I
 4220 M9=I-2
     :REM=============================================REM
*M9 is the no of constraints
```

Program Listing for RPM

```
 4230 F2$=HEX(0182)
      :$UNPACK (D=F2$)A8$()TO D$()
 4240 @@: FOR I=1TO 14
      :<>: IF D$(I)=" " THEN  4250
      :L=L+1
      :C$(L)=D$(I)
 4250 ↑↑: NEXT I
      :INIT(20)D$()
      :<>: IF STR(G8$(2),1,8)=" " THEN  4260
      :A0$=STR(G8$(2),1,8)
      :GOSUB '40
      :<-: GOTO  4230
 4260 REM=============================================REM
******* create primal and dual graph ***********
      :A0$=B0$
 4270 <<: PRINT "Do you want to clear the data disk";
      :>>: INPUT W$
      :<>: IF W$="Y" THEN  4280
      :<-: GOTO  4300
 4280 @@: FOR I=1TO 20
      :DATA LOAD DA F(I,L7)F0$,F1,F2,F3,F0,F()
      :F$=" "
      :F0,F1,F2,F3=0
      :MAT F=ZER
      :DATA SAVE DA F(I,L6)F0$,F1,F2,F3,F0,F()
      :↑↑: NEXT I
      :L7,L8=0
 4290 @@: FOR I=1TO 20
      :DATA LOAD DA F(I+500,L7)F0$,F1,F2,F3,F0,F()
      :F$=" "
      :F0,F1,F2,F3=0
      :MAT F=ZER
      :DATA SAVE DA F(I+500,L6)F0$,F1,F2,F3,F0,F()
      :↑↑: NEXT I
      :L7,L8=0
 4300 GOSUB '40
      :@@: FOR I0=1TO 14
      :<>: IF STR(A8$(I0),1,3)<>"COL" THEN  4310
      :<-: GOTO  4320
 4310 ↑↑: NEXT I0
```

Program Listing for RPM

```
 4320 IO=IO+1
     :N9=N9+1
     :REM=============================================REM
N9-1  is the no. of variables
 4330 F1$=HEX(0120)
     :INIT(20)D$()
     :MAT D=ZER
     :MAT E=ZER
     :<>: IF STR(A8$(IO),1,3)="RHS" THEN  4450
     :$UNPACK (D=F1$)A8$(IO)TO D$()
     :@@: FOR I=3TO M+1STEP 2
     :<>: IF D$(I)=" " THEN  4340
     :K=INT(I/2)
     :CONVERT D$(I)TO D(K)
     :↑↑: NEXT I
 4340 REM=============================================REM

 4350 @@: FOR I=2TO MSTEP 2
     :<>: IF D$(I)=" " THEN  4370
     :@@: FOR J=1TO 20
     :<>: IF D$(I)=STR(B$(J),2,7) THEN  4360
     :↑↑: NEXT J
     :<-: GOTO  4370
 4360 E(I/2)=J
     :NO=NO+1
     :↑↑: NEXT I
 4370 REM=============================================REM

 4380 LO=LO+1
     :F2=0
     :@@: FOR I=1TO M
     :<>: IF E(I)=0 THEN  4400
     :<>: IF E(I)=M9+1 THEN  4390
     :I2=2*I
     :F(I2-1)=E(I)
     :F(I2)=D(I)
     :↑↑: NEXT I
     :<-: GOTO  4400
 4390 F2=D(I)
     :NO=NO-1
```

Program Listing for RPM

```
 4400 FO$=C$(LO)
     :FO=NO
 4410 REM=============================================REM
**
     :GOSUB '42
     :NO=0
     :F2=0
 4420 REM=============================================REM
***E is the resource no.(sector),D is the arrow value.
 4430 <>: IF IO<14 THEN  4320
     :IO=0
     :AO$=STR(G8$(2),1,8)
     :<>: IF AO$=" " THEN  4440
     :GOSUB '40
     :<-: GOTO  4320
 4440 REM=============================================REM

 4450 REM=============================================REM

 4461 GOSUB '46
     :                          M3=N9-1
     :K3=0
     :GOSUB '48
     :<<: PRINT HEX(0E0A);"Process Graph"
     :GOSUB '44(M3,K3)
     :REM=============================================REM
**print dual graph
 4471 M3=M9
     :K3=500
     :<<: PRINT HEX(0E0A);"Resource Graph"
     :GOSUB '44(M3,K3)
     :REM=============================================REM
**print primal graph
 4480 REM=============================================REM

 4490 REM=============================================REM
*******end of the main program **********
     :STOP
  4500 DEFFN '42
     :DATA SAVE DA F(LO,L3)FO$,F1,F2,F3,FO,F()
```

Program Listing for RPM

```
4510 @@: FOR I=1TO NO
     :MAT R=ZER
     :RO=O
     :DATA LOAD DA F(500+E(I),L3)RO$,R1,R2,R3,RO,R()
4520 RO$=8$(E(I))
     :R(2*RO+1)=N9
     :R(2*(RO+1))=D(I)
     :RO=RO+1
4530 DATA SAVE DA F(500+E(I),L3)RO$,R1,R2,R3,RO,R()
     :↑↑: NEXT I
     :RETURN :↑↑:
4540 REM=============================================REM

 4550 DEFFN '40
     :DATA LOAD DC OPEN T#1,AO$
     :DATA LOAD DC #1,G8$(),A8$()
     :RETURN :↑↑:
 4560 DEFFN '41
     :DATA LOAD DA T#1,(KO,MO)G8$(),A8$()
     :RETURN :↑↑:
 4570 DEFFN '44(M3,K3)
     :GOSUB '49
     :L=O
     :FO=O
     :@@: FOR I=1TO M3
     :DATA LOAD DA F(K3+I,L3)FO$,F1,F2,F3,FO,F()
     :<<: PRINTUSING  4590 ,FO$,F1,F2,F3,FO;
     :<>: IF FO=O THEN  4580
     :@@: FOR I5=1TO FO
     :I6=2*I5-1
     :<<: PRINTUSING  4600 ,F(I6+1),F(I6);
     :↑↑: NEXT I5
     :<<: PRINT
4580 ↑↑: NEXT I
     :RETURN :↑↑:
4590 %######### -####.# -####.# -####.#     <##>
4600 % -####.#(##)
 4610 DEFFN '46
     :IO=IO+1
     :F2$=HEX(0120)
```

Program Listing for RPM

```
      :$UNPACK (D=F2$)A8$(10)TO D$()
      :@@: FOR I=2TO M8STEP 2
      :<>: IF D$(I)=" " THEN  4640
 4620 @@: FOR J=1TO M9
      :<>: IF D$(I)=STR(B$(J),2,7) THEN  4630
      :↑↑: NEXT J
 4630 DATA LOAD DA F(500+J,L3)F0$,F1,F2,F3,F0,F()
      :CONVERT D$(I+1)TO F2
      :DATA SAVE DA F(500+J,L3)F0$,F1,F2,F3,F0,F()
      :↑↑: NEXT I
 4640 RETURN :↑↑:
  4650 DEFFN '49
      :REM=========================================REM
***output
 4660 <>: IF M3>500 THEN  4670
      :<<: PRINTUSING  4680
      :<<: PRINT
      :RETURN :↑↑:
 4670 <<: PRINTUSING  4690
      :<<: PRINT
      :RETURN :↑↑:
 4680  %Resource  Value  Constant  Residue   Arcs   =====
===>
 4690  %Process   Value  Constant  Residue   Arcs   =====
===>
  4700 DEFFN '48
      :SELECT <<: PRINT 215(110)
      :<<: PRINT HEX(0A0A)
      :<<: PRINT HEX(0E);TAB(20);"RPM NETWORK "
      :<<: PRINT HEX(0E0A);TAB(4);"Title
      :";Q$
      :N9=N9-1
      :DATA SAVE DA F(1001,L3)M9,N9,Q$
      :<<: PRINT
      :RETURN :↑↑:
```

End of RPM Program Listing

Program Listing for NETALG

```
 1000 REM=============================================REM
*NETWORK* (07/30/80)GDS
 1005  SELECT <<: PRINT 215(114)
 1010 DIM F(10),K(5),U(5),S(14),S1(14),B(14),B1(14),Y(14)
 1011 DIM Y0(14),Y1(14),I(14),I$(14)1
 1020 DIM R(10),H(5),V(5),D(14),D1(14),C(14),C1(14),X(14)
 1021 DIM X0(14),X1(14),J(14),J$(14)1
 1030 DIM L(11,11),Q(11,11),P$1
 1040 W=10↑10
 1060 REM=============================================REM
  ==== resource data ==================
 1070 REM=============================================REM
  = K  - primal node number           =
 1080 REM=============================================REM
  = U  - arc from resource to process =
 1090 REM=============================================REM
  = B  - resource input               =
 1100 REM=============================================REM
  = B1 - resource buffer              =
 1110 REM=============================================REM
  = S  - resource residue             =
 1120 REM=============================================REM
  = Y  - resource variable            =
 1130 REM=============================================REM
  = Y1 - resource variable buffer     =
 1140 REM=============================================REM
  = Y0 - resource variable buffer     =
 1150 REM=============================================REM
  ==== process data ===================
 1160 REM=============================================REM
  = H  - resource node number         =
 1170 REM=============================================REM
  = V  - arc from process to resource =
 1180 REM=============================================REM
  = C  - cost coefficient             =
 1190 REM=============================================REM
  = C1 - cost coefficient buffer      =
 1200 REM=============================================REM
  = D1 - process residue              =
```

Program Listing for NETALG

```
1210 REM=============================================REM
 = X  - process variable                 =
1220 REM=============================================REM
 = X1 - process variable buffer          =
1230 REM=============================================REM
 = XO - process variable buffer          =
1240 REM=============================================REM

1270 MAT XO=ZER
     :MAT X1=ZER
     :MAT X=ZER
     :MAT B=ZER
     :MAT B1=ZER
     :MAT I=ZER
     :INIT(20)I$()
1280 MAT YO=ZER
     :MAT Y1=ZER
     :MAT Y=ZER
     :MAT C=ZER
     :MAT C1=ZER
     :MAT J=ZER
     :INIT(20)J$()
1290 MAT L=ZER
     :MAT Q=ZER
1320 REM=============================================REM
              load residues
1332 DIM Q$45
     :DATA LOAD DA F(1001,L3)M,N,Q$
     :I9=M
     :J9=N
1333 @@: FOR J=1TO N
     :GOSUB '50(J)
     :C(J)=C
     :D(J)=-C
     :↑↑: NEXT J
1334 @@: FOR I=1TO M
     :GOSUB '51(I)
     :B(I)=B
     :S(I)=B
     :↑↑: NEXT I
```

Program Listing for NETALG

```
 1336 GOSUB '80
 1351 REM=========================================REM

 1370 REM=========================================REM
              Check for negative process residue
 1380 @@: FOR J=1TO N
     :<>: IF D(J)>=0 THEN  1381
     :J1=J
     :GOSUB '52
     :<->: GOTO  1500
 1381 ↑↑: NEXT J
 1390 REM=========================================REM
              Check for negative resource residue
 1400 @@: FOR I=1TO M
     :<>: IF S(I)>=0 THEN  1401
     :I1=I
     :GOSUB '53
     :<->: GOTO  1500
 1401 ↑↑: NEXT I
 1440 REM=========================================REM
              Check for negative variables
 1470 @@: FOR I=1TO M
     :<>: IF Y(I)>=0 THEN  1471
     :I1=I
     :GOSUB '54
     :<->: GOTO  1500
 1471 ↑↑: NEXT I
 1480 @@: FOR J=1TO N
     :<>: IF X(J)>=0 THEN  1481
     :J1=J
     :GOSUB '74
     :<->: GOTO  1500
 1481 ↑↑: NEXT J
 1490 <<: PRINT "Current Solution is Optimum "
     :STOP
 1500 <>: ON F<->: GOTO  1510 , 1520 , 1530 , 1540
     :STOP
```

Program Listing for NETALG

```
 1510 GOSUB '57
     :<-: GOTO  1550
 1520 GOSUB '58
     :<-: GOTO  1550
 1530 GOSUB '59
     :<-: GOTO  1550
 1540 GOSUB '60
     :<-: GOTO  1550
 1550 GOSUB  4230
     :GOSUB  4240
     :GOSUB '55
     :GOSUB '56
     :GOSUB '63
     :GOSUB '64
     :GOSUB '61
     :GOSUB '62
     :GOSUB '80
 1590 REM=============================================REM

 1600 <-: GOTO  1351
 1610 REM=============================================REM

 1700 STOP
 1710 REM=============================================REM

 1730 REM=============================================REM

 1740 REM=============================================REM
              S U B R O U T I N E S
 1760 REM=============================================REM
              Load Process Node   -50-
 1780   DEFFN '50(K3)
     :REM=============================================REM
*K3 is the process number
 1790 MAT H=ZER
     :MAT V=ZER
     :DATA LOAD DA F(K3,L3)F0$,F1,C,A,T,F()
     :@@: FOR L9=1TO T
```

Program Listing for NETALG

```
     :H(L9)=F(2*L9-1)
     :V(L9)=F(2*L9)
     :↑↑: NEXT L9
     :RETURN :↑↑:
 1800 REM=============================================REM

 1820 REM=============================================REM
               Load Resource Node    -51-
  1840    DEFFN  '51(K3)
     :REM=============================================REM
*K3 is the resource number
 1850 MAT K=ZER
     :MAT U=ZER
     :DATA LOAD DA F(500+K3,L3)F0$,F1,B,A,T,F()
     :@@: FOR L9=1TO T
     :K(L9)=F(2*L9-1)
     :U(L9)=F(2*L9)
     :↑↑: NEXT L9
     :RETURN :↑↑:
 1860 REM=============================================REM

 1880 REM=============================================REM
               Type (a) Labeling     -52-
  1900  DEFFN  '52
     :T5=0
     :MAT B1=ZER
     :MAT S1=ZER
 1910  T0=0
     :T1=W
     :T2=W
     :P$=" "
     :GOSUB '50(J1)
 1920 T0=T0+1
     :<>: IF T0>T THEN  1950
     :<>: IF I$(H(T0))="*" THEN  1940
     :<-: GOTO  1920
 1940 P$="*"
     :GOSUB  4200
     :B1(I)=-V(T0)
```

Program Listing for NETALG

```
     :<-: GOTO  1920
1950 <>: IF P$=" " THEN  1970
     :GOSUB '55
     :J=0
1960  J=J+1
     :<>: IF J>R THEN  1970
     :J5=J(J)
     :<>: IF X1(J)>=0 THEN  1960
     :T5=-X(J5)/X1(J)
     :J0=J5
     :<>: IF T5>T2 THEN  1960
     :T1=T5
     :J2=J5
     :<-: GOTO  1960
1970 X1(J1)=1
     :J$(J1)="*"
     : @@: FOR I=1TO M
     :<>: IF I$(I)="*" THEN  1975
     :<>: IF S(I)<0 THEN  1975
     :S=0
     :GOSUB '51(I)
     :@@: FOR K=1TO T
     :K1=K(K)
     :<>: IF J$(K1)=" " THEN  1972
     :S=S-X1(K1)*U(K)
1972 ↑↑: NEXT K
     :S1(I)=S
     :<>: IF S>=0 THEN  1975
     :T5=-S(I)/S
     :I0=I
     :<>: IF T5>=T2 THEN  1975
     :T2=T5
     :I1=I0
1975 ↑↑: NEXT I
     :<>: IF T1+T2=2*W THEN  1983
     :<>: IF T1<=T2 THEN  1982
1981 I$(I1)="*"
     :J$(J1)="*"
     :F=1
     :RETURN :↑↑:
```

Program Listing for NETALG

```
1982 J$(J1)="*"
    :J$(J2)=" "
    :F=2
    :RETURN :↑↑:
1983 <<: PRINT
    :<<: PRINT "Solution is unbounded"
    :STOP
1999 REM=============================================REM
              Type (b) Labeling     -53-
2000 DEFFN '53
    :T5=0
    :MAT C1=ZER
    :MAT D1=ZER
2010  T0=0
    :T1=W
    :T2=W
    :T3=W
    :P$=" "
    :GOSUB '51(I1)
2020 T0=T0+1
    :<>: IF T0>T THEN  2050
    :<>: IF J$(K(T0))="*" THEN  2040
    :<-: GOTO  2020
2040 P$="*"
    :GOSUB  4210
    :C1(J)=-U(T0)
    :<-: GOTO  2020
2050 <>: IF P$=" " THEN  2070
    :GOSUB '56
    :I=0
2060  I=I+1
    :<>: IF I>R THEN  2070
    :I5=I(I)
    :<>: IF Y1(I)>=0 THEN  2060
    :T5=-Y(I5)/Y1(I)
    :I0=I5
    :<>: IF T5>T2 THEN  2060
    :T1=T5
    :I2=I5
```

Program Listing for NETALG

```
    :<-: GOTO  2060
2070 Y1(I1)=1
    :I$(I1)="*"
    : @@: FOR J=1TO N
    :<>: IF J$(J)="*" THEN  2075
    :<>: IF D(J)<0 THEN  2075
    :D=0
    :GOSUB '50(J)
    :@@: FOR H=1TO T
    :H1=H(H)
    :<>: IF I$(H1)=" " THEN  2072
    :D=D-Y1(H1)*V(H)
2072 ↑↑: NEXT H
    :D1(J)=-D
    :<>: IF D1(J)>=0 THEN  2075
    :T5=-D(J)/D1(J)
    :J0=J
    :<>: IF T5>=T2 THEN  2075
    :T2=T5
    :J1=J0
2075 ↑↑: NEXT J
    :<>: IF T1+T2=2*W THEN  2083
    :<>: IF T1<=T2 THEN  2082
2081 J$(J1)="*"
    :I$(I1)="*"
    :F=1
    :RETURN :↑↑:
2082 I$(I1)="*"
    :I$(I2)=" "
    :F=2
    :RETURN :↑↑:
2083 <<: PRINT
    :<<: PRINT "Solution is infeasible"
    :STOP
2180 REM=============================================REM
              Type (c) Labeling    -54-
 2200   DEFFN '54
    :MAT B1=ZER
    :MAT S1=ZER
221^  T0=0
```

Program Listing for NETALG

```
     :T1=W
     :T2=W
     :T=W
     :T5=0
     :@@: FOR I=1TO R
     :<>: IF I(I)<>I1 THEN  2211
     :B1(I)=-1
     :<-: GOTO  2220
2211 ↑↑: NEXT I
2220 GOSUB '55
     :J=0
2230  J=J+1
     :<>: IF J>R THEN  2240
     :J5=J(J)
     :<>: IF X1(J)>=0 THEN  2230
     :T5=-X(J5)/X1(J)
     :J0=J5
     :<>: IF T5>T2 THEN  2230
     :T1=T5
     :J1=J5
     :<-: GOTO  2230
2240 @@: FOR I=1TO M
     :<>: IF I$(I)="*" THEN  2245
     :<>: IF S(I)<0 THEN  2245
     :S=0
     :GOSUB '51(I)
     :@@: FOR K=1TO T
     :K1=K(K)
     :<>: IF J$(K1)=" " THEN  2242
     :S=S-X1(K1)*U(K)
2242 ↑↑: NEXT K
     :S1(I)=S
     :<>: IF S>=0 THEN  2245
     :T5=-S(I)/S
     :I0=I
     :<>: IF T5>=T2 THEN  2245
     :T2=T5
     :I2=I0
2245 ↑↑: NEXT I
     :<>: IF T1<=T2 THEN  2248
```

Program Listing for NETALG

```
2246 I$(I1)=" "
     :I$(I2)="*"
     :F=3
     :RETURN :↑↑:
2248 I$(I1)=" "
     :J$(J1)=" "
     :F=4
     :RETURN :↑↑:
2250 REM=============================================REM
              Type (c2) Labeling     -74-
2252    DEFFN '74
     :MAT C1=ZER
     :MAT D1=ZER
2254  T0=0
     :T1=W
     :T2=W
     :T=W
     :T5=0
     :@@: FOR J=1TO R
     :<>: IF J(J)<>J1 THEN  2256
     :C1(J)=-1
     :<-: GOTO  2258
2256 ↑↑: NEXT J
2258 GOSUB '56
     :I=0
2260  I=I+1
     :<>: IF I>R THEN  2262
     :I5=I(I)
     :<>: IF Y1(I)>=0 THEN  2260
     :T5=-Y(I5)/Y1(I)
     :I0=J5
     :<>: IF T5>T2 THEN  2260
     :T1=T5
     :I1=I5
     :<-: GOTO  2260
2262 @@: FOR J=1TO N
     :<>: IF J$(J)="*" THEN  2266
     :<>: IF D(J)<0 THEN  2266
     :D=0
     :GOSUB '50(J)
```

Program Listing for NETALG

```
     :@@: FOR H=1TO T
     :H1=H(H)
     :<>: IF I$(H1)=" " THEN  2264
     :D=D-Y1(H1)*V(H)
2264 ↑↑: NEXT H
     :D1(J)=D
     :<>: IF D>=0 THEN  2266
     :T5=-D(J)/D
     :J0=J
     :<>: IF T5>=T2 THEN  2266
     :T2=T5
     :J2=J0
2266 ↑↑: NEXT J
     :<>: IF T1<=T2 THEN  2270
2268 J$(J1)=" "
     :J$(J2)="*"
     :F=3
     :RETURN :↑↑:
2270 J$(J1)=" "
     :I$(I1)=" "
     :F=4
     :RETURN :↑↑:
2290  REM=============================================REM
           Primal Scanning      -55-
2300  DEFFN '55
     :MAT X0=ZER
     :MAT X1=ZER
     :S=0
2301 <>: IF R>1 THEN  2302
     :X1(1)=B1(1)/Q(1,1)
     :RETURN :↑↑:
2302 REM=============================================REM

2310 @@: FOR I=1TO R
     :S=0
     :@@: FOR K=1TO I-1
     :S=S+L(I,K)*X0(K)
     :↑↑: NEXT K
     :X0(I)=B1(I)-S
     :↑↑: NEXT I
```

Program Listing for NETALG

```
2315 @@: FOR I=1TO R
    :S=0
    :<>: IF I=1 THEN  2316
    :@@: FOR K=1TO I-1
    :S=S+Q(R+1-I,R+1-K)*X1(R+1-K)
    :↑↑: NEXT K
2316 L=R+1-I
    :X1(L)=(XO(L)-S)/Q(L,L)
    :↑↑: NEXT I
2330 RETURN :↑↑:
2390  REM==========================================REM
          Dual Scanning        -56-
2400  DEFFN '56
    :MAT YO=ZER
    :MAT Y1=ZER
    :S=0
2401 <>: IF R>1 THEN  2402
    :Y1(1)=C1(1)/Q(1,1)
    :RETURN :↑↑:
2402 REM==========================================REM

2410 @@: FOR J=1TO R
    :S=0
    :@@: FOR K=1TO J-1
    :S=S+Q(K,J)*YO(K)
    :↑↑: NEXT K
    :YO(J)=(1/Q(J,J))*(C1(J)-S)
    :↑↑: NEXT J
2415 @@: FOR J=RTO 1STEP -1
    :S=0
    :<>: IF J=R THEN  2416
    :@@: FOR K=J+1TO R
    :S=S+L(K,J)*Y1(K)
    :↑↑: NEXT K
2416 Y1(J)=YO(J)-S
    :↑↑: NEXT J
2430 RETURN :↑↑:
2610  REM==========================================REM
        Type 1 Factoring   -57-
2630  DEFFN '57
```

Program Listing for NETALG

```
     :MAT B1=ZER
     :MAT C1=ZER
     :V1=0
2640  GOSUB '51(I1)
     :@@: FOR I=1TO T
     :<>: IF K(I)<>J1 THEN  2641
     :V1=U(I)
2641 @@: FOR J=1TO R
     :<>: IF K(I)<>J(J) THEN  2642
     :C1(J)=U(I)
2642 ↑↑: NEXT J
     :↑↑: NEXT I
2658  GOSUB '50(J1)
     :@@: FOR J=1TO T
     :@@: FOR I=1TO R
     :<>: IF H(J)<>I(I) THEN  2659
     :B1(I)=V(J)
2659 ↑↑: NEXT I
     :↑↑: NEXT J
     :<>: IF R>0 THEN  2660
     :L(1,1)=1
     :Q(1,1)=V1
     :<-: GOTO  2696
2660 <>: IF R>1 THEN  2661
     :L(2,1)=C1(1)/Q(1,1)
     :L(2,2)=1
     :Q(1,2)=B1(1)
     :Q(2,2)=V1-L(2,1)*Q(1,2)
     :<-: GOTO  2696
2661 L(R+1,1)=C1(1)/Q(1,1)
     :@@: FOR J=2TO R
     :S=0
     :@@: FOR K=1TO J-1
     :S=S+L(R+1,K)*Q(K,J)
     :↑↑: NEXT K
     :L(R+1,J)=(C1(J)-S)/Q(J,J)
     :↑↑: NEXT J
     :L(R+1,R+1)=1
2662 B1(R+1)=V1
2665 Q(1,R+1)=B1(1)
```

Program Listing for NETALG

```
     :@@: FOR I=2TO R+1
     :S=0
     :@@: FOR K=1TO I-1
     :S=S+L(I,K)*Q(K,R+1)
     :↑↑: NEXT K
     :Q(I,R+1)=(B1(I)-S)
     :↑↑: NEXT I
2696 R=R+1
     :I(R)=I1
     :J(R)=J1
     :RETURN :↑↑:
2697 REM=============================================REM

2710  REM=============================================REM
        Type 2a Factoring   -58-
2730   DEFFN '58
     :MAT B1=ZER
     :MAT C1=ZER
     :GOSUB  4220
     :J3=J
     :<>: IF J3<R THEN  2740
     :@@: FOR I=1TO Q
     :Q(I,R)=0
     :↑↑: NEXT I
     :<-: GOTO  2758
2740 @@: FOR J=J3TO R-1
     :J(J)=J(J+1)
     :@@: FOR I=1TO R
     :Q(I,J)=Q(I,J+1)
     :Q(I,R)=0
     :↑↑: NEXT I
     :↑↑: NEXT J
     :J(R)=J1
2750 @@: FOR J=J3TO R-1
     :<>: IF Q(J,J)=0 THEN  2790
     :T=Q(J+1,J)/Q(J,J)
     :Q(J+1,J)=0
     :L(J+1,J)=L(J+1,J)+L(J+1,J+1)*T
     :Q(J+1,J+1)=T*Q(J,J+1)+Q(J+1,J+1)
```

Program Listing for NETALG

```
     :↑↑: NEXT J
 2758  GOSUB '50(J1)
     :@@: FOR J=1TO T
     :@@: FOR I=1TO R
     :<>: IF H(J)<>I(I) THEN  2759
     :B1(I)=V(J)
 2759 ↑↑: NEXT I
     :↑↑: NEXT J
 2760 @@: FOR I=1TO R
     :S=0
     :<>: IF I>1 THEN  2770
     :Q(1,R)=B1(1)
     :↑↑: NEXT I
     :<-: GOTO  2780
 2770 @@: FOR K=1TO I-1
     :S=S+L(I,K)*Q(K,R)
     :↑↑: NEXT K
     :Q(I,R)=B1(I)-S
     :↑↑: NEXT I
 2780 J(R)=J1
     :RETURN :↑↑:
 2790 INIT(20)I$(),J$()
     :MAT L=ZER
     :MAT Q=ZER
     :R1=R
     :R=0
     :@@: FOR L6=1TO R1
     :I$(I(L6))="*"
     :J$(J(L6))="*"
     :I1=I(L6)
     :J1=J(L6)
     :GOSUB '57
     :↑↑: NEXT L6
     :RETURN :↑↑:
 2810  REM=============================================REM
          Type 2b Factoring   -59-
 2830   DEFFN '59
     :MAT B1=ZER
     :MAT C1=ZER
 2840 @@: FOR I=I1TO R-1
```

Program Listing for NETALG

```
    :@@: FOR J=1TO R
    :L(J,I)=L(J,I+1)
    :L(J,R)=0
    :↑↑: NEXT J
    :↑↑: NEXT I
2850 @@: FOR I=I1TO R-1
    :T=L(I+1,I)/L(I,I)
    :L(I+1,I)=0
    :Q(I+1,I)=Q(I+1,I)+Q(I+1,I+1)*T
    :L(I+1,I+1)=T*L(I,I+1)+L(I+1,I+1)
    :↑↑: NEXT I
2858  GOSUB '51(I2)
    :@@: FOR I=1TO T
    :@@: FOR J=1TO R
    :<>: IF K(I)<>J(J) THEN  2859
    :C1(J)=U(I)
2859 ↑↑: NEXT J
    :↑↑: NEXT I
2860 @@: FOR J=1TO R
    :S=0
    :<>: IF J>1 THEN  2770
    :L(1,R)=C1(1)
    :↑↑: NEXT J
    :<-: GOTO  2880
2870 @@: FOR K=1TO J-1
    :S=S+Q(J,K)*L(K,R)
    :↑↑: NEXT K
    :Q(J,R)=C1(J)-S
    :↑↑: NEXT J
2880 I(R)=I2
    :RETURN :↑↑:
2891  REM==========================================REM
        Type 3 Factoring   -60-
2893 DEFFN '60
    :@@: FOR L=1TO R
    :<>: IF I(L)<>I1 THEN  2894
    :L9=L
    :@@: FOR L9=LTO R-1
    :I(L9)=I(L9+1)
    :↑↑: NEXT L9
```

Program Listing for NETALG

```
     :I(R)=0
     :<-: GOTO  2895
 2894 ↑↑: NEXT L
     :STOP "2894"
 2895 @@: FOR L=1TO R
     :<>: IF J(L)<>J1 THEN  2896
     :L9=L
     :@@: FOR L9=LTO R-1
     :J(L9)=J(L9+1)
     :↑↑: NEXT L9
     :J(R)=0
     :<-: GOTO  2899
 2896 ↑↑: NEXT L
 2899 INIT(20)I$(),J$()
     :MAT L=ZER
     :MAT Q=ZER
     :R1=R-1
     :R=0
     :@@: FOR L6=1TO R1
     :I$(I(L6))="*"
     :J$(J(L6))="*"
     :I1=I(L6)
     :J1=J(L6)
     :GOSUB '57
     :R=R+1
     :↑↑: NEXT L6
     :RETURN :↑↑:
 3010 REM=============================================REM
          Resource Balancing  -61-
 3021 REM=============================================REM

 3030 DEFFN '61
 3040  @@: FOR I=1TO M
     :S=0
     :<>: IF I$(I)<>"*" THEN  3050
     :S(I)=0
     :↑↑: NEXT I
     :RETURN :↑↑:
 3050  GOSUB '51(I)
     :@@: FOR K=1TO T
```

Program Listing for NETALG

```
     :K1=K(K)
     :<>: IF J$(K1)=" " THEN  3060
     :S=S+X(K1)*U(K)
3060  ↑↑: NEXT K
     :S(I)=B(I)-S
     :↑↑: NEXT I
     :RETURN :↑↑:
3080 REM=============================================REM
          Process Balancing  -62-
3090 DEFFN '62
4000 @@: FOR J=1TO N
     :D=0
     :<>: IF J$(J)<>"*" THEN  4010
     :D(J)=0
     :↑↑: NEXT J
     :RETURN :↑↑:
4010 GOSUB '50(J)
     :@@: FOR H=1TO T
     :H1=H(H)
     :<>: IF I$(H1)=" " THEN  4020
     :D=D+Y(H1)*V(H)
4020  ↑↑: NEXT H
     :D(J)=-C(J)+D
     :↑↑: NEXT J
     :RETURN :↑↑:
4021 REM=============================================REM

4110 REM=============================================REM
          Update Process Val.-63-
4125 DEFFN '63
     :MAT X=ZER
4130 @@: FOR J=1TO R
     :X(J(J))=X1(J)
     :↑↑: NEXT J
     :RETURN :↑↑:
4131 REM=============================================REM

4150 REM=============================================REM
          Update Resour. Val.-64-
4156 DEFFN '64
```

Program Listing for NETALG

```
    :MAT Y=ZER
4157 @@: FOR I=1TO R
    :Y(I(I))=Y1(I)
    :↑↑: NEXT I
    :RETURN :↑↑:
4162 @@: FOR J=1TO R
    :X(J(J))=X1(J)
    :↑↑: NEXT J
    :RETURN :↑↑:
4170 @@: FOR I=1TO R
    :Y(I(I))=Y1(I)
    :↑↑: NEXT I
    :RETURN :↑↑:
4200 @@: FOR I=1TO R
    :<>: IF I(I)=H(TO) THEN  4299
    :↑↑: NEXT I
    :STOP "4200"
4210 @@: FOR J=1TO R
    :<>: IF J(J)=K(TO) THEN  4299
    :↑↑: NEXT J
    :STOP "4210"
4220 @@: FOR J=1TO R
    :<>: IF J(J)=J2 THEN  4299
    :↑↑: NEXT J
    :STOP "4220"
4230 @@: FOR I=1TO R
    :B1(I)=B(I(I))
    :↑↑: NEXT I
    :RETURN :↑↑:
4240 @@: FOR J=1TO R
    :C1(J)=C(J(J))
    :↑↑: NEXT J
    :RETURN :↑↑:
4299 RETURN :↑↑:
5001 REM==========================================REM
       P r i n t i n g   S u b r o u t i n e s
5005 DEFFN '80
    :<<: PRINT HEX(0C0E0A0A);"Linear RPM Networks"
    :<<: PRINT HEX(0E0A0A);"     ";Q$
    :<<: PRINT
```

Program Listing for NETALG

```
     :<<: PRINT "Iteration No. ";O;
     :O=O+1
     :Z=O
     :@@: FOR J=1TO N
     :Z=Z+C(J)*X(J)
     :↑↑: NEXT J
     :<<: PRINTUSING  5007 ,Z
     :<<: PRINT
 5006 <<: PRINT "Resource Graph"
     :<<: PRINTUSING  5101 ," ";
     :@@: FOR I=1TO 4
     :<<: PRINTUSING  5111 ," ";
     :↑↑: NEXT I
     :<<: PRINT
 5007 %         Objective Function -###,###,###.###
 5010 @@: FOR I=1TO M
     :GOSUB '51(I)
     :<<: PRINTUSING  5100 ,I,FO$,I$(I),Y(I),B(I),S(I),T;
     :@@: FOR K=1TO T
     :<<: PRINTUSING  5110 ,U(K),K(K);
     :↑↑: NEXT K
     :<<: PRINT
     :↑↑: NEXT I
     :<<: PRINT
 5100 %## ######## # -#######.### -#######.### -#######.###    #
 5101 %No  Name  St        Value      Constant       Resid.   N#
 5110 %-####.##(##)
 5111 % Arc(Node)#
 5205 <<: PRINT "Process Graph"
 5210 @@: FOR J=1TO N
     :GOSUB '50(J)
     :<<: PRINTUSING  5100 ,J,FO$,J$(J),X(J),C(J),D(J),T;
     :@@: FOR H=1TO T
     :<<: PRINTUSING  5110 ,V(H),H(H);
     :↑↑: NEXT H
     :<<: PRINT
     :↑↑: NEXT J
     :<<: PRINT
 7776 <<: PRINT "===  labeled resource network ==="
 7777 @@: FOR I=1TO R
```

Program Listing for NETALG

```
     :@@: FOR J=1TO R
     :<<: PRINTUSING  7778 ,L(I,J);
     :↑↑: NEXT J
     :<<: PRINT
     :↑↑: NEXT I
     :<<: PRINT
     :<<: PRINT
     :<<: PRINT
 7778 % -###.###
 7779 <<: PRINT "===  labeled process network  ==="
 7787 @@: FOR I=1TO R
     :@@: FOR J=1TO R
     :<<: PRINTUSING  7778 ,Q(I,J);
     :↑↑: NEXT J
     :<<: PRINT
     :↑↑: NEXT I
     :<<: PRINT
     :<<: PRINT
 8001 RETURN :↑↑:
 9998 DEFFN R(X)=SGN(X)*
   INT(ABS(X)*1000+.5)/1000
```

End of NETALG Program Listing

APPENDIX C

DATA FILES

```
Title:     RPM test file  (3X4)
ROWS
,<RES001,<RES002,<RES003,$PROFIT
COLUMNS
PR001 RES001 1 RES002 -2 RES003 3 PROFIT 2
PR002 RES001 3 RES002 -16 RES003 -1 PROFIT 1
PR003 RES001 2 RES002 -1 RES003 -5 PROFIT 4
PR004 RES001 5 RES002 -1 RES003 10 PROFIT 5
RHS
RESOURCE RES001 20 RES002 -4 RES003 -10
END
```

EX1

```
....:....1....:....2....:....3....:....4....:....5....:....6.:
Title:     RPM test file  (3X4)                                  01
ROWS                                                             02
,<RES001,<RES002,<RES003,$PROFIT                                 03
COLUMNS                                                          04
{PR001 RES001 1 RES002 -2 RES003 3 PROFIT 2                      05
{PR002 RES001 3 RES002 -16 RES003 -1 PROFIT 1                    06
{PR003 RES001 2 RES002 -1 RES003 -5 PROFIT 4                     07
{PR004 RES001 5 RES002 -1 RES003 10 PROFIT 5                     08
RHS                                                              09
RESOURCE RES001 20 RES002 -4 RES003 -10                          10
END                                                              11
                                                                 12
                                                                 13
                                                                 14
....:....1....:....2....:....3....:....4....:....5....:....6.:
```

```
Title:     RPM test file   (3X4)
ROWS
,<RES001,<RES002,<RES003,$PROFIT
COLUMNS
PRO01 RES001 -9 RES002 1 RES003 -7 PROFIT 4
PRO02 RES001 -5 RES002 -1 RES003 3 PROFIT -3
PRO03 RES001 -1 RES002 3 RES003 3 PROFIT 2
PRO04 RES001 6 RES002 3 RES003 8 PROFIT 1
RHS
RESOURCE RES001 2 RES002 10 RES003 0
END
```

EX3

```
....:....1....:....2....:....3....:....4....:....5....:....6.:
Title:     RPM test file   (3X4)                                  01
ROWS                                                              02
,<RES001,<RES002,<RES003,$PROFIT                                  03
COLUMNS                                                           04
{PRO01 RES001 -9 RES002 1 RES003 -7 PROFIT 4                      05
{PRO02 RES001 -5 RES002 -1 RES003 3 PROFIT -3                     06
{PRO03 RES001 -1 RES002 3 RES003 3 PROFIT 2                       07
{PRO04 RES001 6 RES002 3 RES003 8 PROFIT 1                        08
RHS                                                               09
RESOURCE RES001 2 RES002 10 RES003 0                              10
END                                                               11
                                                                  12
                                                                  13
                                                                  14
....:....1....:....2....:....3....:....4....:....5....:....6.:
```

```
Title:     RPM test file  Example #6
ROWS
,<RES001,<RES002,<RES003,$PROFIT
COLUMNS
PRO01 RES001 -2 RES002 3 RES003 -1 PROFIT 2
PRO02 RES001 3 RES002 -4 RES003 -2 PROFIT 5
PRO03 RES001 -1 RES002 -5 RES003 7 PROFIT 6
PRO04 RES001 -4 RES002 -1 RES003 -8 PROFIT 1
PRO05 RES001 -5 RES002 -9 RES003 5 PROFIT 3
END
RESOURCE RES001 -10 RES002 -5 RES003 -8
END
```

EX14

```
....:....1....:....2....:....3....:....4....:....5....:....6.:
Title:     RPM test file  Example #6                          01
ROWS                                                          02
,<RES001,<RES002,<RES003,$PROFIT                              03
COLUMNS                                                       04
!PRO01 RES001 -2 RES002 3 RES003 -1 PROFIT 2                  05
!PRO02 RES001 3 RES002 -4 RES003 -2 PROFIT 5                  06
!PRO03 RES001 -1 RES002 -5 RES003 7 PROFIT 6                  07
!PRO04 RES001 -4 RES002 -1 RES003 -8 PROFIT 1                 08
!PRO05 RES001 -5 RES002 -9 RES003 5 PROFIT 3                  09
END                                                           10
RESOURCE RES001 -10 RES002 -5 RES003 -8                       11
END                                                           12
                                                              13
                                                              14
....:....1....:....2....:....3....:....4....:....5....:....6.:
```

```
Title:     RPM test data file
ROWS
,<RES001,<RES002,<RES003,<RES004,<RES005,<RES006
,<RES007,<RES008,<RES009,<RES010,$PROFIT
COLUMNS
PR001 RES001 1 RES004 -.6 RES005 -.5 RES006 -.6 PROFIT -500
PR002 RES002 1 RES004 -.7 RES005 -.5 RES006 -.3 PROFIT -40
PR003 RES003 1 RES004 -.5 RES005 -.4 RES006 -.5 PROFIT -70
PR004 RES004 1 RES007 -1 RES008 -1 RES009 -1
PR005 RES005 1 RES007 -1 RES008 -1 RES009 -1
PR006 RES006 1 RES007 -1 RES008 -1 RES009 -1
PR007 RES007 1 RES010 -1
PR008 RES008 1 RES010 -1
PR009 RES009 1 RES010 -1
PR010 RES007 1 RES010 -1
PR011 RES008 1 RES010 -1
PR012 RES009 1 PROFIT 50
PR013 RES010 1 PROFIT 200
RHS
RESOURCE RES001 60 RES002 60 RES003 90
END
```

RPM2

```
....:....1....:....2....:....3....:....4....:....5....:....6.:
{PR010 RES007 1 RES010 -1                                      01
{PR011 RES008 1 RES010 -1                                      02
{PR012 RES009 1 PROFIT 50                                      03
{PR013 RES010 1 PROFIT 200                                     04
RHS                                                            05
RESOURCE RES001 60 RES002 60 RES003 90                         06
END
```

RPM1

```
....:....1....:....2....:....3....:....4....:....5....:....6.:
Title:     RPM test data file                                  01
ROWS                                                           02
,<RES001,<RES002,<RES003,<RES004,<RES005,<RES006               03
,<RES007,<RES008,<RES009,<RES010,$PROFIT                       04
COLUMNS                                                        05
{PR001 RES001 1 RES004 -.6 RES005 -.5 RES006 -.6 PROFIT -500   06
{PR002 RES002 1 RES004 -.7 RES005 -.5 RES006 -.3 PROFIT -40    07
....:....1...  {PR003 RES003 1 RES004 -.5 RES005 -.4 RES006 -.5 PROFIT -70   08
{PR004 RES004 1 RES007 -1 RES008 -1 RES009 -1                  09
{PR005 RES005 1 RES007 -1 RES008 -1 RES009 -1                  10
{PR006 RES006 1 RES007 -1 RES008 -1 RES009 -1                  11
{PR007 RES007 1 RES010 -1                                      12
{PR008 RES008 1 RES010 -1                                      13
{PR009 RES009 1 RES010 -1                                      14
....:....1....:....2....:....3....:....4....:....5....:....6.:
```

APPENDIX D

SAMPLE RUNS

RPM test file

Iteration No. 2 Objective Function 40.000

Resource Graph No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	2.000	20.000	0.000	4	1.0 (1)	3.0 (2)	2.0 (3)	5.0 (4)
2	<RES002		0.000	-4.000	16.909	4	-2.0 (1)	-16.0 (2)	-1.0 (3)	-1.0 (4)
3	<RES003	*	0.000	-10.000	0.000	4	3.0 (1)	-1.0 (2)	-5.0 (3)	10.0 (4)
Process Graph										
1	PRO01	*	7.272	2.000	0.000	3	1.0 (1)	-2.0 (2)	3.0 (3)	
2	PRO02		0.000	1.000	5.000	3	3.0 (1)	-16.0 (2)	-1.0 (3)	
3	PRO03	*	6.363	4.000	0.000	3	2.0 (1)	-1.0 (2)	-5.0 (3)	
4	PRO04		0.000	5.000	5.000	3	5.0 (1)	-1.0 (2)	10.0 (3)	

Linear RPM Networks

RPM test file

Iteration No. 3 Objective Function 4.368

Resource Graph No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	0.026	7.000	0.000	4	8.0 (1)	3.0 (2)	4.0 (3)	1.0 (4)
2	<RES002	*	1.394	3.000	0.000	4	2.0 (1)	6.0 (2)	1.0 (3)	5.0 (4)
3	<RES003		0.000	8.000	6.631	4	1.0 (1)	4.0 (2)	5.0 (3)	2.0 (4)
Process Graph										
1	PRO01	*	0.842	3.000	0.000	3	8.0 (1)	2.0 (2)	1.0 (3)	
2	PRO02		0.000	4.000	4.447	3	3.0 (1)	6.0 (2)	4.0 (3)	
3	PRO03		0.000	1.000	0.499	3	4.0 (1)	1.0 (2)	5.0 (3)	
4	PRO04	*	0.263	7.000	0.000	3	1.0 (1)	5.0 (2)	2.0 (3)	

Linear RPM Networks

RPM test file

Iteration No. 1 Objective Function 40.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	2.000	92.000	4	-9.00(1)	-5.00(2)	-1.00(3)	6.00(4)
2	<RES002	*	4.000	10.000	0.000	4	1.00(1)	-1.00(2)	3.00(3)	3.00(4)
3	<RES003		0.000	0.000	70.000	4	-7.00(1)	3.00(2)	3.00(3)	8.00(4)

Process Graph

1	PRO01	*	10.000	4.000	0.000	3	-9.00(1)	1.00(2)	-7.00(3)
2	PRO02		0.000	-3.000	-1.000	3	-5.00(1)	-1.00(2)	3.00(3)
3	PRO03		0.000	2.000	10.000	3	-1.00(1)	3.00(2)	3.00(3)
4	PRO04		0.000	1.000	11.000	3	6.00(1)	3.00(2)	8.00(3)

Solution is unbounded

Linear RPM Networks

RPM test file

Iteration No. 3 Objective Function 32.837

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	0.186	13.000	0.000	2	1.0 (1)	4.0 (2)		
2	<RES002	*	0.651	45.000	0.000	2	12.0 (1)	5.0 (2)		
3	<RES003		0.000	16.000	1.860	2	5.0 (1)	1.0 (2)		

Process Graph

1	PRO01	*	1.976	8.000	0.000	3	1.0 (1)	12.0 (2)	5.0 (3)
2	PRO02	*	4.255	4.000	0.000	3	4.0 (1)	5.0 (2)	1.0 (3)

RPM test file

Iteration No. 4 Objective Function 113.571

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	4.714	15.000	0.000	4	1.0 (1)	1.0 (2)	1.0 (3)	1.0 (4)
2	<RES002		0.000	90.000	16.428	4	7.0 (1)	5.0 (2)	3.0 (3)	2.0 (4)
3	<RES003	*	0.428	100.000	0.000	4	3.0 (1)	4.0 (2)	10.0 (3)	8.0 (4)

Process Graph

1	PR001	*	7.142	6.000	0.000	3	1.0 (1)	7.0 (2)	3.0 (3)	
2	PR002		0.000	5.100	1.328	3	1.0 (1)	5.0 (2)	4.0 (3)	
3	PR003	*	7.857	9.000	0.000	3	1.0 (1)	3.0 (2)	10.0 (3)	
4	PR004		0.000	6.000	2.142	3	1.0 (1)	2.0 (2)	8.0 (3)	

Linear RPM Networks

RPM test file

Iteration No. 2 Objective Function 26.500

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	6.000	0.500	1	1.0 (1)			
2	<RES002		0.000	8.000	12.000	2	-1.0 (1)	1.0 (2)		
3	<RES003	*	3.250	7.000	0.000	2	1.0 (1)	1.0 (2)		
4	<RES004	*	0.243	15.000	0.000	2	3.0 (1)	-1.0 (2)		

Process Graph

1	PR001	*	5.500	4.000	0.000	4	1.0 (1)	-1.0 (2)	1.0 (3)	3.0 (4)
2	PR002	*	1.500	3.000	0.000	3	1.0 (2)	1.0 (3)	-1.0 (4)	

Linear RPM Networks

RPM test file

Iteration No. 2 Objective Function 21.333

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	4.000	3.066	2	0.1 (1)	0.2 (2)		
2	<RES002	*	0.066	50.000	0.000	1	30.0 (2)			
3	<RES003	*	3.000	6.000	0.000	1	1.0 (1)			
4	<RES004		0.000	-0.500	134.166	2	-18.0 (1)	-16.0 (2)		
5	<RES005		0.000	-0.100	85.566	2	-14.0 (1)	-1.0 (2)		

Process Graph

1	PR001	*	6.000	3.000	0.000	4	0.1 (1)	1.0 (3)	-18.0 (4)	-14.0 (5)
2	PR002	*	1.666	2.000	0.000	4	0.2 (1)	30.0 (2)	-16.0 (4)	-1.0 (5)

Linear RPM Networks

RPM test file

Iteration No. 2 Objective Function -966,666.666

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	2000.000	0.000	3	3.0 (1)	1.0 (2)	6.0 (3)	
2	<RES002	*	333.333	-2000.000	0.000	3	-3.0 (1)	-1.0 (2)	-6.0 (3)	
3	<RES003	*	300.000	-1000.000	0.000	3	-2.0 (1)	-5.0 (2)	-1.0 (3)	
4	<RES004		0.000	3000.000	2111.111	3	1.0 (1)	2.0 (2)	4.0 (3)	

Process Graph

1	PR001	*	444.444	-1600.000	0.000	4	3.0 (1)	-3.0 (2)	-2.0 (3)	1.0 (4)
2	PR002		0.000	-3200.000	1366.666	4	1.0 (1)	-1.0 (2)	-5.0 (3)	2.0 (4)
3	PR003	*	111.111	-2300.000	0.000	4	6.0 (1)	-6.0 (2)	-1.0 (3)	4.0 (4)

RPM test file

Iteration No. 1 Objective Function 18.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	5.000	0.500	3	3.0 (1)	-2.0 (2)	1.0 (3)	
2	<RES002	*	6.000	3.000	0.000	3	2.0 (1)	1.0 (2)	-1.0 (3)	
3	<RES003		0.000	15.000	1.500	3	9.0 (1)	-6.0 (2)	3.0 (3)	

Process Graph

1	PR001	*	1.500	12.000	0.000	3	3.0 (1)	2.0 (2)	9.0 (3)	
2	PR002		0.000	8.000	-2.000	3	-2.0 (1)	1.0 (2)	-6.0 (3)	
3	PR003		0.000	4.000	-10.000	3	1.0 (1)	-1.0 (2)	3.0 (3)	

Solution is unbounded

RPM test data file

Iteration No. 5 Objective Function 60.607

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	1.250	10.000	0.000	2	4.0 (2)	3.0 (3)		
2	<RES002	*	0.607	15.000	0.000	1	7.0 (1)			
3	<RES003	*	0.749	12.000	0.000	2	5.0 (1)	3.0 (3)		
4	<RES004	*	3.000	7.000	0.000	3	3.0 (4)	1.0 (5)	2.0 (6)	
5	<RES005	*	1.000	9.000	0.000	1	4.0 (5)			
6	<RES006		0.000	0.000	3.369	3	-3.0 (2)	2.0 (4)	3.0 (6)	

Process Graph

1	PRO01	*	2.142	8.000	0.000	2	7.0 (2)	5.0 (3)		
2	PRO02	*	2.178	5.000	0.000	2	4.0 (1)	-3.0 (6)		
3	PRO03	*	0.428	6.000	0.000	2	3.0 (1)	3.0 (3)		
4	PRO04	*	1.583	9.000	0.000	2	3.0 (4)	2.0 (6)		
5	PRO05	*	2.250	7.000	0.000	2	1.0 (4)	4.0 (5)		
6	PRO06		0.000	5.000	1.000	2	2.0 (4)	3.0 (6)		

Linear RPM Networks

RPM test file 2X2

Iteration No. 2 Objective Function 63.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	2.250	18.000	0.000	2	3.0 (1)	2.0 (2)		
2	<RES002	*	2.249	10.000	0.000	2	1.0 (1)	2.0 (2)		

Process Graph

1	PRO01	*	3.999	9.000	0.000	2	3.0 (1)	1.0 (2)		
2	PRO02	*	3.000	9.000	0.000	2	2.0 (1)	2.0 (2)		

Current Solution is Optimum

RPM test data file

Iteration No. 5 Objective Function 47.027

Resource Graph No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	0.977	10.000	0.000	3	2.0 (1)	4.0 (2)	3.0 (3)	
2	<RES002	*	0.863	15.000	0.000	3	7.0 (1)	3.0 (2)	6.0 (3)	
3	<RES003		0.000	12.000	5.181	2	5.0 (1)	3.0 (3)		
4	<RES004	*	1.800	7.000	0.000	3	3.0 (4)	1.0 (5)	2.0 (6)	
5	<RES005	*	1.300	9.000	0.000	3	2.0 (4)	4.0 (5)	3.0 (6)	
6	<RES006	*	0.500	0.000	0.000	3	-3.0 (2)	2.0 (4)	3.0 (6)	

Process Graph										
1	PRO01	*	1.363	8.000	0.000	3	2.0 (1)	7.0 (2)	5.0 (3)	
2	PRO02	*	1.818	5.000	0.000	3	4.0 (1)	3.0 (2)	-3.0 (6)	
3	PRO03		0.000	6.000	2.113	3	3.0 (1)	6.0 (2)	3.0 (3)	
4	PRO04	*	1.486	9.000	0.000	3	3.0 (4)	2.0 (5)	2.0 (6)	
5	PRO05	*	0.886	7.000	0.000	2	1.0 (4)	4.0 (5)		
6	PRO06	*	0.827	9.000	0.000	3	2.0 (4)	3.0 (5)	3.0 (6)	

Linear RPM Networks

RPM test file

Iteration No. 2 Objective Function 20.000

Resource Graph No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	4.000	5.000	0.000	3	3.0 (1)	-2.0 (2)	1.0 (3)	
2	<RES002	*	0.000	3.000	0.000	3	2.0 (1)	1.0 (2)	-1.0 (3)	
3	<RES003		0.000	15.000	0.000	3	9.0 (1)	-6.0 (2)	3.0 (3)	

Process Graph										
1	PRO01	*	1.600	12.000	0.000	3	3.0 (1)	2.0 (2)	9.0 (3)	
2	PRO02		0.000	-8.000	0.000	3	-2.0 (1)	1.0 (2)	-6.0 (3)	
3	PRO03	*	0.200	4.000	0.000	3	1.0 (1)	-1.0 (2)	3.0 (3)	

RPM test data file

Iteration No. 11 Objective Function 5,700.000

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	40.000	20.000	0.000	1	1.0 (1)			
2	<RES002	*	150.000	30.000	0.000	1	1.0 (2)			
3	<RES003	*	40.000	10.000	0.000	1	1.0 (3)			
4	<RES004	*	200.000	0.000	0.000	4	-0.2 (1)	-0.4 (2)	-0.1 (3)	1.0 (4)
5	<RES005	*	200.000	0.000	0.000	4	-0.2 (1)	-0.2 (2)	-0.1 (3)	1.0 (5)
6	<RES006	*	200.000	0.000	0.000	4	-0.3 (1)	-0.4 (2)	-0.2 (3)	1.0 (6)
7	<RES007	*	200.000	0.000	0.000	3	-1.0 (4)	1.0 (7)	1.0 (10)	
8	<RES008	*	200.000	0.000	0.000	3	-1.0 (5)	1.0 (8)	1.0 (11)	
9	<RES009	*	200.000	0.000	0.000	3	-1.0 (6)	1.0 (9)	1.0 (12)	
10	<RES010	*	200.000	0.000	0.000	4	-1.0 (7)	-1.0 (8)	-1.0 (9)	1.0 (13)
11	<RES011	*	80.000	0.000	0.000	4	-1.0 (10)	-1.0 (11)	-1.0 (12)	1.0 (14)

Process Graph

1	PR001	*	20.000	-100.000	0.000	4	1.0 (1)	-0.2 (4)	-0.2 (5)	-0.3 (6)
2	PR002	*	30.000	-50.000	0.000	4	1.0 (2)	-0.4 (4)	-0.2 (5)	-0.4 (6)
3	PR003	*	10.000	-40.000	0.000	4	1.0 (3)	-0.1 (4)	-0.1 (5)	-0.2 (6)
4	PR004	*	17.000	0.000	0.000	2	1.0 (4)	-1.0 (7)		
5	PR005	*	11.000	0.000	0.000	2	1.0 (5)	-1.0 (8)		
6	PR006	*	20.000	0.000	0.000	2	1.0 (6)	-1.0 (9)		
7	PR007	*	17.000	0.000	0.000	2	1.0 (7)	-1.0 (10)		
8	PR008	*	11.000	0.000	0.000	2	1.0 (8)	-1.0 (10)		
9	PR009	*	20.000	0.000	0.000	2	1.0 (9)	-1.0 (10)		
10	PR010		0.000	0.000	120.000	2	1.0 (7)	-1.0 (11)		
11	PR011		0.000	0.000	120.000	2	1.0 (8)	-1.0 (11)		
12	PR012		0.000	0.000	120.000	2	1.0 (9)	-1.0 (11)		
13	PR013	*	48.000	200.000	0.000	1	1.0 (10)			
14	PR014	*	0.000	80.000	0.000	1	1.0 (11)			

RPM test data file

Iteration No. 8 Objective Function 2.582

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001		0.000	120.000	119.593	1	1.0 (4)			
2	<RES002		0.000	48.000	48.000	1	1.0 (5)			
3	<RES003	*	152.582	0.000	0.000	3	-1.0 (3)	1.0 (6)	1.0 (7)	
4	<RES004	*	43.250	0.000	0.000	4	-1.0 (4)	-1.0 (5)	0.4 (6)	0.5 (7)
5	<RES005	*	26.550	0.000	0.000	2	-6.4 (6)	1.0 (8)		
6	<RES006	*	17.415	0.000	0.000	2	-8.5 (7)	1.0 (10)		
7	<RES007		0.000	120.000	119.557	1	1.0 (1)			
8	<RES008		0.000	48.000	48.000	1	1.0 (2)			
9	<RES009	*	50.000	0.000	0.000	4	-1.0 (1)	-1.0 (2)	0.0 (8)	0.0 (10)
10	<RES010	*	30.000	0.000	0.000	2	-1.0 (8)	1.0 (9)		
11	<RES011	*	20.000	0.000	0.000	2	-1.0 (10)	1.0 (11)		
12	<RES012	*	0.005	500.000	0.000	1	500.0 (3)			

Process Graph

1	PRO01	*	0.440	-50.000	0.000	2	1.0 (7)	-1.0 (9)		
2	PRO02		0.000	-75.000	25.000	2	1.0 (8)	-1.0 (9)		
3	PRO03	*	1.000	-150.000	0.000	2	-1.0 (3)	500.0 (12)		
4	PRO04	*	0.410	-43.250	0.000	2	1.0 (1)	-1.0 (4)		
5	PRO05		0.000	-64.880	21.630	2	1.0 (2)	-1.0 (4)		
6	PRO06	*	1.000	0.000	0.000	3	1.0 (3)	0.4 (4)	-6.4 (5)	
7	PRO07		0.000	0.000	26.090	3	1.0 (3)	0.5 (4)	-8.5 (6)	
8	PRO08	*	6.410	0.000	0.000	3	1.0 (5)	0.0 (9)	-1.0 (10)	
9	PRO09	*	6.410	30.000	0.000	1	1.0 (10)			
10	PRO10	*	0.000	0.000	0.000	3	1.0 (6)	0.0 (9)	-1.0 (11)	
11	PRO11	*	0.000	20.000	0.000	1	1.0 (11)			

RPM test data file

Iteration No. 5 Objective Function 61.700

Resource Graph

No	Name	St	Value	Constant	Resid.	N	Arc(Node)	Arc(Node)	Arc(Node)	Arc(Node)
1	<RES001	*	1.250	10.000	0.000	2	4.0 (2)	3.0 (3)		
2	<RES002		0.000	15.000	12.600	1	1.0 (1)			
3	<RES003	*	1.600	12.000	0.000	2	5.0 (1)	3.0 (3)		
4	<RES004	*	3.000	7.000	0.000	3	3.0 (4)	1.0 (5)	2.0 (6)	
5	<RES005	*	1.000	9.000	0.000	1	4.0 (5)			
6	<RES006		0.000	0.000	4.333	4	-3.0 (2)	2.0 (4)	3.0 (6)	-3.0 (9)
7	<RES007	*	2.000	0.000	0.000	4	3.0 (7)	5.0 (8)	3.0 (9)	2.0 (10)
8	<RES008		0.000	0.000	0.000	4	4.0 (7)	3.0 (8)	2.0 (9)	4.0 (10)

Process Graph

1	PR001	*	2.400	8.000	0.000	2	1.0 (2)	5.0 (3)		
2	PR002	*	2.500	5.000	0.000	2	4.0 (1)	-3.0 (6)		
3	PR003		0.000	6.000	2.550	2	3.0 (1)	3.0 (3)		
4	PR004	*	1.583	9.000	0.000	2	3.0 (4)	2.0 (6)		
5	PR005	*	2.250	7.000	0.000	2	1.0 (4)	4.0 (5)		
6	PR006		0.000	0.000	6.000	2	2.0 (4)	3.0 (6)		
7	PR007		0.000	0.000	6.000	2	3.0 (7)	4.0 (8)		
8	PR008	*	0.000	10.000	0.000	2	5.0 (7)	3.0 (8)		
9	PR009		0.000	0.000	6.000	3	3.0 (7)	2.0 (8)	-3.0 (6)	
10	PR010		0.000	3.000	1.000	2	2.0 (7)	4.0 (8)		