DTIC FILE COPY

2

AD-A198 429

RADC-TR-88-17
Final Technical Report
February 1988

NETWORK MANAGEMENT OF HIGHLY ADAPTIVE COMMUNICATION NETWORKS

Southeastern Center for Electrical Engineering Education

Jeffery L. Kennington, Richard V. Helgason and John M. Colombi, 2Lt, USAF

DTIC
ELECTE
AUG 2 6 1988
S D
E

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded totally by the Laboratory Directors' fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

88 8 25 194

RADC-TR-88-17 has been reviewed and is approved for publication.

APPROVED: [illegible]

JOHN M. COLOMBI, 2Lt, USAF
Project Engineer

APPROVED: [illegible]

BRUNO BEEK
Technical Director
Directorate of Communications

FOR THE COMMANDER: [illegible]

JAMES W. HYDE, III
Directorate of Plans & Programs

DESTRUCTION NOTICE - For classified documents, follow the procedures in DOD 5200.22-M, Industrial Security Manual, or DOD 5200.1-R, Information Security Program Regulation. For unclassified limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCLD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE		Form Approved OMB No 0704-0188
1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS: N/A	
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A		
4. PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S): RADC-TR-88-17	
6a. NAME OF PERFORMING ORGANIZATION: Southeastern Center for Electrical Engineering Education	6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION: Rome Air Development Center (DCLD)
6c. ADDRESS *(City, State, and ZIP Code)*: Central Florida Facility, 1101 Massachusetts Avenue, St. Cloud FL 32769		7b. ADDRESS *(City, State, and ZIP Code)*: Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center	8b OFFICE SYMBOL *(If applicable)*: DCLD	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F30602-81-C-0193
8c. ADDRESS *(City, State, and ZIP Code)*: Griffiss AFB NY 13441-5700		10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
61101F	LDFP	15	P7

11. TITLE *(Include Security Classification)*
NETWORK MANAGEMENT OF HIGHLY ADAPTIVE COMMUNICATION NETWORKS

12 PERSONAL AUTHOR(S)
Jeffrey L. Kennington, Richard V. Helgason, John M. Colombi, 2Lt, USAF

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Final	FROM Aug 86 TO Jul 87	February 1988	128

16. SUPPLEMENTARY NOTATION
This effort was funded totally by the Laboratory Directors' fund.

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Operations Research, Communications Networks, Parallel Processing, Network Flow Problems, Linear Programming
17	02		

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
This report documents networking models, network solutions, programming techniques for parallel processing, and parallelized algorithm comparisons. Several papers are contained in the report. An operational research model and associated mathematics are presented for a three node network. A multi-media nodal simulation is developed to optimally assign trunks. A new mathematical approach is shown for solving equal flow problems. This technique makes greater use of the side constraints structure with computational solutions given. Also developed are the mathematical theory and justification of using the quadrant interlocking factorization for solving the simplex algorithm on a parallel processor. Lastly, computational results of solving minimal spanning tree problems, on a parallel processor are presented.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE *(Include Area Code)*	22c. OFFICE SYMBOL
John M. Colombi, 2Lt, USAF	(315) 330-7751	RADC (DCLD)

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ACKNOWLEDGEMENT

This research was supported in part by the Rome Air Development Center under contract number F30603-81-0193. Funding was made available through the Laboratory Director's Fund Program.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

1 INTRODUCTION

This report details the work accomplished under the US Air Force, Rome Air Development Center Contract No. F30602-81-C-0193. The twelve month Post Doctoral effort used as a foundation the education of the Department of Operations Research, School of Engineering and Applied Sciences at Southern Methodist University, Dallas, Texas. The principal investigator was Dr. Jeffery L. Kennington.

The objective was to take optimization theory developed by the Air Force Office of Scientific Research for solving multi commodity problems, and apply it to the topic of communication network management. Under Project Forecast II (PFII), a concept for a highly adaptive communication network was developed. This complex and survivable network will require new techniques for managing its available resources. These results will further develop classes of algorithms that can manage this network. The results contained are a compilation of five separate reports, contained in Appendices A - E.

2 SCOPE

The structure of this report not only reflects the work done, but the evolution of this work, as well. Initially, efforts concentrated on modelling communication networks and later progressed into algorithm development for high performance computer architectures.

Appendix A summarizes an initial OR model for a three node network. The solution makes use of a commodity based supply and demand model.

Appendix B is a new technique for handling network flow solutions. In particular, the method is a means for solving linear equal flow problems, by more efficiency utilizing the special structure of the side constraints. Computational solution time results are given.

Appendix C summarizes the development of an optimization model for a single communications node having multiple media. This code provides an initial capability for examining node managment and its functions. Incorporated into the model are capacity and delay constraints.

Appendix D is a Ph.D. dissertation of Dr. Hossam Zaki who was supported on this effort. This report presents the quadrant interlocking factorization for solving the simplex algorithm for linear problems on parallel machines. Included in the report are the relevant justifications and algorithms required for implementation.

Appendix E contains computational results of work done on a twenty-CPU Balance 21000 computer. Algorithms for solving minimal spanning trees were investigated. Comparisons of sequential and parallel solutions for these algorithms are presented.

3 CONCLUSIONS

As presented in many of the references, much work is being accomplished in OR techniques for a variety of applications. Many more communications network management functions and networking problems are being near optimally handled.

The next major efficiency increase appears to be found in more powerful computer architectures with new "supercomputing" algorithms. One specific application which needs development is communication algorithms for parallel processing machines, and the last two reports are examplary work in this area. Initial summaries reflect that a possible sub-optimal algorithm may prove more efficient for use on a multiple CPU architecture.

Under the Project Forecast II Program, classes of network algorithms are needed to manage the PF II type of network. This network may incorporate parallel processors to perform a variety of user and resource managment functions. This work and further developments will feed into the design and implementation of the specific algorithms to be incorporated in the Project Forecast II 62702F Program.

APPENDIX A

AN OPERATIONS RESEARCH MODEL

R. V. Helgason

J. L. Kennington

Department of Operations Research
Southern Methodist University
Dallas, Texas 75275

This report gives an example of an adaptive communication network and an operations research model which could be used to solve it. Consider the 3 node communication network given in Figure 1.

Suppose lines 1,3,5,6,7 are telephone lines and 2 and 4 are micro wave links. Suppose the message table is:

From \ To	1	2	3	Totals
1	-	5	2	7
2	1	-	0	1
3	4	7	-	11
Totals	5	12	2	

That is, 5 messages must be sent from node 1 to node 2 during the next time period.

We model this using 3 commoditites (one for each origin node). Then the supply and demands are as follows:

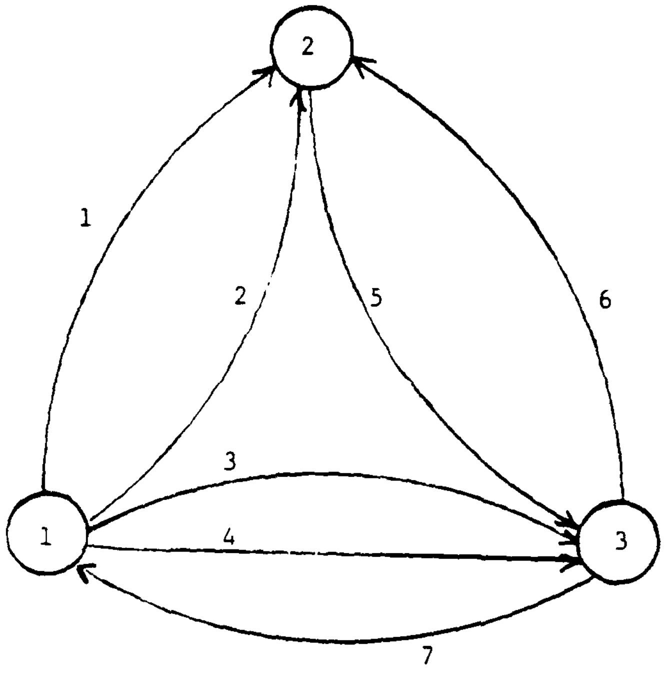


Figure 1

Commodity	1	2	3
Supply			
Node 1	7	0	0
Node 2	0	1	0
Node 3	0	0	11
Demand			
Node 1	0	1	4
Node 2	5	0	7
Node 3	2	0	0
Requirement (Supply - Demand)			
Node 1	7	-1	-4
Node 2	-5	1	-7
Node 3	-2	0	11

Suppose that node 1 can communicate with node 2 using either telephone lines or micro wave links, but not both and that node 1 can communicate with node 3 using either telephone lines or micro wave links, but not both. Then the management decisions are to determine which edges to use and how many messages should be assigned to each link. Note that messages going from 1 to 2 can go from 1 to 3 and then from 3 to 2

We now define a mathematical model corresponding to this example.

<u>Subscripts</u>

k - denotes the commodity (origin node).

i - denotes the node.

j - denotes the edge.

<u>Constants</u>

c_j^k - denotes the cost of sending one message originating at k along j.

u_j - denotes the capacity of edge j.

A - denotes the node-arc incidence matrix corresponding to the graph.

M - denotes the sum of all supplies for all commodities.

For this example:

$$A = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{c} \text{(edges)} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{(node 1)} \\ \text{(node 2)} \\ \text{(node 3).} \end{array}$$

$M = 19.$

r_i^k - denotes the requirement at node i of commodity k.

r^k - denotes the vector of requirements for commodity k.

For this example:

$$r^1 = \begin{bmatrix} 7 \\ -5 \\ -2 \end{bmatrix}, \quad r^2 = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \quad r^3 = \begin{bmatrix} -4 \\ -7 \\ 11 \end{bmatrix}.$$

f_j - denotes the fixed cost for using edge j.

n - denotes the number of edges.

m - denotes the number of nodes.

<u>Decision Variables</u>

y_j = 0, if edge j is not used and 1 otherwise.

x_j^k - denotes the flow of commodity k on edge j.

The Model

minimize $\sum_{k=1}^{m} \sum_{j=1}^{n} c_j^k x_j^k + \sum_{j=1}^{n} f_j y_j$ (1)

subject to: $Ax^k = r^k \quad , k = 1,\ldots m$ (2)

$\sum_{k=1}^{n} x_j^k \leq u_j \; , j = 1,\ldots,n$ (3)

$\sum_{k=1}^{n} x_j^k \leq My_j, \; j = 1,\ldots,n$ (4)

$y_1 + y_2 \leq 1$ (5)

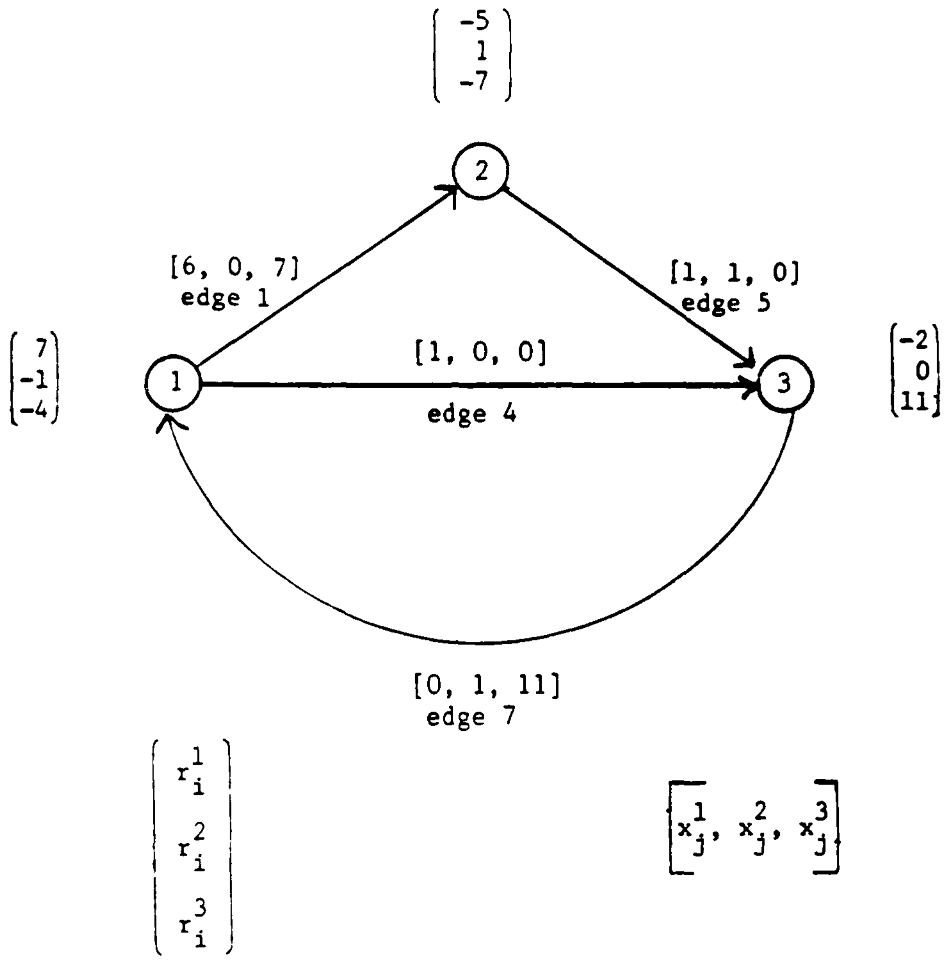
$y_3 + y_4 \leq 1$ (6)

$x_j^k \geq 0$, all j,k (7)

$y_j \in \{0, 1\}$, all j . (8)

Constraints (2) ensure that the messages begin and terminate at the correct nodes. These are sometimes called flow conservation constraints. Constraints (3) ensure that the capacity (bandwidth) of the edges is not exceeded. Constraints (4) ensure that $y_j = 1$ if any messages are assigned to edge j. Constraint (5) ensures that at most one of the edges 1 and 2 are used. Constraints (7) ensure that messages travel in the proper direction.

A feasible solution to this prcblem may be:



$\begin{pmatrix} -5 \\ 1 \\ -7 \end{pmatrix}$
2
[6, 0, 7] edge 1
[1, 1, 0] edge 5
$\begin{pmatrix} 7 \\ -1 \\ -4 \end{pmatrix}$
1
[1, 0, 0]
edge 4
3
$\begin{pmatrix} -2 \\ 0 \\ 11 \end{pmatrix}$
[0, 1, 11] edge 7
$\begin{pmatrix} r_i^1 \\ r_i^2 \\ r_i^3 \end{pmatrix}$
$[x_j^1, x_j^2, x_j^3]$

Note that 2 messages were transmitted from node 1 to node 3. One used edges 1 and 5 and the other used edge 4.

APPENDIX B

THE LINEAR EQUAL FLOW PROBLEM

Agha Iqbal Ali

Department of General Business
The University of Texas at Austin

Jeff Kennington

Department of Operations Research
Southern Methodist University

Bala Shetty

Department of Business Analysis
Texas AM University

ACKNOWLEDGEMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278, the Department of Defense under Contract Number MDA 903-86-C-0182, and Rome Air Development Center under Contract Number SCEEE PDP/86-75.

PREFACE

This paper presents a new algorithm for the solution of a network problem with equal flow side constraints. The solution technique is motivated by the desire to exploit the special structure of the side constraints and to maintain as much of the characteristics of pure network problems as possible. The proposed algorithm solves the equal flow problem using two sequences of pure network problems. One sequence corresponds to computing a lower bound while the other corresponds to computing an upper bound. Step sizes exist such that both bounds converge to the optimal objective value. Termination when the difference between the bounds is within a prespecified tolerance is a particularly attractive feature of the solution procedure employed. The algorithm has been tested on problems with up to 1500 nodes and 6000 arcs. Computational experience indicates that feasible solutions whose objective function value is within 10% of the optimum can be obtained in 6% of the time required for MPSX to obtain an optimum. Guaranteed 5% solutions can be obtained in 17% of the MPSX time.

I. INTRODUCTION

This paper presents a new technique to solve the linear equal flow problem. The problem is easily conceptualized as a minimal cost network flow problem with additional constraints on certain pairs of arcs. Specifically, given pairs of arcs are required to take on the same value. The problem is defined on a network represented by an m x n node-arc incidence matrix, A, in which K pairs of arcs are identified and required to have equal flow. Mathematically, this is expressed as:

$$\begin{aligned} &\text{Minimize } \mathbf{cx} \\ &\text{s.t.} \quad \mathbf{Ax} = \mathbf{b} \\ &\qquad x_k = x_{k+K}, \quad k = 1,2,\ldots,K \\ &\qquad \mathbf{0} \le \mathbf{x} \le \mathbf{u} \end{aligned}$$

where **c** is a 1 x n vector of unit costs, **b** is an m x 1 vector of node requirements, **0** is an n x 1 vector of zeroes, **x** is an n x 1 vector of decision variables, and **u** is an n x 1 vector of upper bounds. This mathematical statement of the problem, henceforth referred to as problem P1, assumes that the first 2K arcs appear in the equal flow constraints. This is not a restrictive assumption, since by rearranging the order of the arcs, any equal flow problem with K pairs can be expressed in the above form. Note that the K pairs of arcs are mutually exclusive, i. e., an arc appears in at most one side constraint. We also assume without loss of generality, that $u_k = u_{k+K}$ for $k = 1,2,\ldots,K$.

When the flow in arcs must be integral, the problem is referred to as an integer equal flow problem. Applications of the integer model include crew scheduling [5], estimating driver costs for transit operations [14], and the two duty period scheduling problem [11]. The linear equal flow problem is a natural relaxation for the integer problem and also provides an approximation to the integer model. The linear model is applicable to problems where integrality is not restrictive. For example, in federal matching of funds allocated to various projects [4].

The linear equal flow problem may be solved using a specialization of the simplex method for networks with side constaints [3]. It has also been solved by transformation to a nonlinear programming problem [4]. By exploiting the special structure of the side constraints and the network structure, this paper develops a new algorithm which results in a decrease in both computer storage and computation time The procedure employs relaxation and decomposition and solves the equal flow problem using two sequences of pure network problems, totally eliminating the computational overhead associated with maintaining a basis matrix. The computational efficiencies in specialized software for the solution of pure networks also extend to the solution of sequences of minimum cost network flow problems by using reoptimization procedures. The reoptimization procedures are used in solving the subproblems of the two sequences.

The use of relaxation techniques and/or decomposition techniques in the solution of problems with special structure in the constraint set is motivated by potential computational efficiencies. Glover, Glover and Martinson [6] address a generalized network problem in which arcs in specified subsets must have proportional flow. The solution approach is via solution of a series of problem relaxations and progressive bound adjustment. The underlying principle is shared in the ensuing development for the equal flow problem.

Lagrangean relaxation has been used to aid in the solution of the integer equal flow problem in two specific instances. Shepardson and Marsten [11] reformulate the two duty period scheduling problem as a single duty period scheduling problem with equal flow side constraints and integrality constraints on the variables. Turnquist and Malandraki [14] model the problem of estimating driver costs for transit operations as an integer equal flow problem. In both studies, the side constraints are dualized and the Lagrangean dual solved using subgradient optimization to yield a lower bound on the optimal objective value. In [14] step-size determination during the subgradient optimization process is aided by a line search. The Lagrangean relaxation, of course, does not enforce the equal flow constraints. The Lagrangean dual for the linear or the integer equal flow problem is exactly the same, since the constraint set for the Lagrangean relaxation is identical. This Lagrangean dual is similar to the quadratic programming problem used in [4]. The similarity lies in penalizing the violating equal flow constraints. The penalty chosen in the quadratic problem formulation should be sufficiently large to guarantee convergence to the optimal solution.

The objective of this investigation is to develop and computationally test a new algorithm for the linear equal flow problem. The solution technique consists of solving two sequences of pure network problems. One sequence progressively yields tighter lower bounds on the optimal value by using the Lagrangean relaxation of the equal flow problem with the side constraints dualized. The second sequence progressively yields upper bounds on the optimal value for the problem and maintains a feasible solution at all times. This sequence is obtained by use of a decomposition of the equal flow problem based on parametric changes in the requirements vector. The solution procedure has the added attractive feature that it provides a feasible solution which is known to be within a percentage of the optimal at all times. As such, the algorithm terminates when a solution with a prespecified tolerance on the objective function value is obtained.

The solution technique makes use of subgradient optimization in the solution of the lower and the upper bounding problems. Both the lower and upper bounding algorithms have been developed in the context of the general subgradient algorithm which is briefly presented in Section II. Section III introduces the Lagrangean dual for the equal flow problem and the lower bounding algorithm. Section IV presents the decomposition of the equal flow problem and the upper bounding algorithm. The overall procedure which makes use of the algorithms of Sections III and IV is given in Section V, computational results are given in Section VI and conclusions drawn in Section VII.

III. THE LOWER BOUND

A lower bound on the objective function of the linear equal flow problem, P1, can be obtained by using the Lagrangean dual of the problem. The lower bound is used in the step size determination as well as in termination criteria for the upper bound procedure. Associating the Lagrange multiplier w_k with the kth equal flow constraint and defining the K-vector $\mathbf{w} = (w_1, w_2, \ldots, w_K)$, the Lagrangean dual for P1, referred to as problem D1, may be stated as

$$\text{maximize } h(w)$$
$$\mathbf{w} \in R^K$$

where $h(w) = \min\{cx + \Sigma_k w_k(x_k - x_{K+k}) \mid Ax = b, 0 \leq x \leq u\}$. Since P1 is a linear program, it is easily established that the optimal objective values of P1 and D1 are equal and that any feasible solution to D1 provides a lower bound on the optimal objective value for P1. For any given value of the vector w, the Lagrangean relaxation is a pure network problem. The subgradient of h at a point $\bar{w}$ is given by the K-vector

$$(\bar{x}_1 - \bar{x}_{K+1}, \ldots, \bar{x}_K - \bar{x}_{2K})$$

where $\bar{x}$ solves the Lagrangean relaxation at $\bar{w}$, given by

$$\{\min cx + \Sigma_k \bar{w}_k(x_k - x_{K+k}) \mid Ax = b, 0 \leq x \leq u\}.$$

ALGORITHM 1 assumed the function f(y) to be convex, whereas h(w) is piece-wise linear concave. The lower bounding algorithm, ALGORITHM 2, modifies the framework of the previous algorithm for a concave function. The step sizes used are given by $\lambda_0 = p$, and $\lambda_i = \lambda_{i-1}/2$. The algorithm makes use of a scalar, UBND, representing an upper bound for the problem. Since the solution procedure progressively improves both the lower bound and the upper bound for the equal flow problem, each time the lower bound algorithm is invoked the value for UBND is obtained from the upper bound procedure. For this algorithm, we assume that both bounds are greater than zero.

II. THE SUBGRADIENT ALGORITHM

The subgradient algorithm was first introduced by Shor [13] and provides a framework for solving nonlinear programming problems. It may be viewed as a generalization of the steepest descent (ascent) method for convex (concave) problems in which the gradient may not exist at all points. At points at which the gradient does not exist, the direction of movement is given by a subgradient. Subgradients do not necessarily provide improving directions and consequently, the convergence results of Zangwill [15] do not apply. Convergence of the subgradient algorithm is assured, however, under fairly minor conditions on the step size.

Given the nonlinear program P0,

$$\begin{aligned} &\text{Minimize} \quad f(y) \\ &\text{s.t.} \qquad y \in G \end{aligned}$$

where f is a real-valued function that is convex over the compact, convex, and nonempty set G, a vector η is a *subgradient* of f at y' if $f(y) - f(y') \geq \eta(y - y')$ for all $y \in G$. For any given $y' \in G$, the set of all subgradients of f at y' is denoted by $\partial f(y')$. Moving a sufficiently large distance s along $-\eta$ can yield a point $x = y' - s\eta$ such that $x \notin G$. The projection of the point x onto G, denoted by $P[x]$, is defined to be the unique point $y \in G$ that is nearest to x with respect to the Euclidean norm. Using the projection operation, the subgradient algorithm in its most general form follows:

ALGORITHM 1: SUBGRADIENT OPTMIZATION ALGORITHM

```
0 Initialization
    Let $y^0 \in G$,
    Select a set of step sizes $s_0, s_1, s_2, \ldots$
    $i \leftarrow 0$.
1 Find Subgradient
    Let $\eta_i \in \partial f(y^i)$.
    If $\eta_i = 0$, then terminate with $y^i$ optimal.
2 Move to new point
    $y^{i+1} \leftarrow P[y^i - s_i \eta_i]$
    $i \leftarrow i + 1$, and return to step 1.
```

There are three general schema which can be used in determining the step size when the subgradient algorithm is implemented for a specific problem:

i. $s_i = \lambda_i$

ii. $s_i = \lambda_i / \|\eta_i\|^2$

iii. $s_i = \lambda_i (f(y^i) - F) / \|\eta_i\|^2$

where F is an estimate of f^*, the optimal value of f over G. A summary of the known convergence results for this algorithm may be found in [2] and [10].

ALGORITHM 2: LOWER BOUND ALGORITHM

```
1 Initialization
    Initialize UBND, step size p, and tolerance $\varepsilon$.
    $w \leftarrow 0$.
2 Find Subgradient
    Let $\bar{x}$ solve $h(w) = \min\{cx + \Sigma_k w_k(x_k - x_{K+k}) \mid Ax = b, 0 \leq x \leq u\}$.
    LBND $\leftarrow h(w)$.
    If (UBND - LBND) $\leq \varepsilon$(UBND), terminate; otherwise,
    $d \leftarrow (\bar{x}_1 - \bar{x}_{K+1}, \ldots, \bar{x}_K - \bar{x}_{2K})$.
3 Move to new point
    (a) $w \leftarrow w + pd$, $p \leftarrow p/2$.
    (b) Go to step 2 .
```

IV. THE UPPER BOUND

An alternate formulation of problem P1, referred to as P2, obtained by decomposing the problem is given by

$$\text{Minimize } g(y)$$
$$\text{s.t. } \quad y \in \mathbf{S}$$

where for any vector $y = (y_1, y_2, \ldots, y_K)$,

$$g(y) = \{\min cx \mid \mathbf{A}x = \mathbf{b};\ \mathbf{0} \le x \le \mathbf{u};\ x_k = x_{K+k} = y_k,\ k = 1,2,\ldots,K\},$$

and,

$$\mathbf{S} = \{y \mid 0 \le y_k \le u_k, \text{ for } k = 1,2,\ldots,K\}.$$

The decomposition assures the satisfaction of the equal flow constraints. The decomposed problem P2 is equivalent to the problem P1 [12] and may be solved using a specialization of the subgradient optimization algorithm. The objective function is piece-wise linear convex and the subgradient η of g at a point y is obtained from the dual variables, v_i, $i = 1,2,\ldots,2K$, associated with the equal flow constraints in the subproblem, referred to as P3 and given by,

$$\begin{array}{llcll}
\text{Minimize} & cx & & & \\
\text{s.t.} & \mathbf{A}x & = & \mathbf{b} & \\
 & x_1 & = & y_1 & (v_1) \\
 & x_{K+1} & = & y_1 & (v_{K+1}) \\
 & \vdots & \vdots & \vdots & \\
 & x_K & = & y_K & (v_K) \\
 & x_{2K} & = & y_K & (v_{2K}) \\
 & \mathbf{0} \le x \le \mathbf{u}. & & &
\end{array}$$

The K-vector

$$\eta = (v_1 + v_{K+1}, v_2 + v_{K+2}, \ldots, v_K + v_{2K})$$

is a *subgradient* of g at $y = (y_1, y_2, \ldots, y_K)$.

The dual variables v_k, $k = 1,2,\ldots,2K$ may easily be constructed from the solution to the pure network problem, referred to as problem P4;

$$\{\min cx \mid Ax = b, \gamma \le x \le \theta\},$$

where the lower and upper bound n-vectors γ and θ are defined by

$$\begin{array}{ll} \gamma_k = \theta_k = y_k, & k = 1,2,\ldots,K \\ \gamma_{K+k} = \theta_{K+k} = y_k, & k = 1,2,\ldots,K \\ \gamma_k = 0,\ \theta_k = u_k, & k = 2K+1,\ldots,n. \end{array}$$

Let Π be the vector of optimal dual variables associated with the conservation of flow constraints, $Ax = b$ in P4 and the arc associated with the variable x_j be incident from node j_f and incident to node j_t. The optimal dual variables for P3 are given by,

$$v_k = -\Pi_{k_f} + \Pi_{k_t} + c_k,\ k = 1,2,\ldots,2K.$$

In using the subgradient optimization algorithm for the decomposed problem at each point y, the subgradient η can be calculated directly using the above development.

It is possible that moving a step along the negative subgradient yields a point which does not belong to the set S. As pointed out in Section II, this point is projected onto the set S by means of a projection operation in the algorithm. For this model, the projection operation decomposes on k so that $P[y] = (P[y_1], P[y_2], \ldots, P[y_K])$ where the projections $P[y_k]$ are defined by:

$$\begin{array}{ll} \text{If } y_k < 0, & P[y_k] = 0. \\ \text{If } y_k > u_k, & P[y_k] = u_k. \\ \text{If } 0 \le y_k \le u_k, & P[y_k] = y_k. \end{array}$$

The subgradient optimization algorithm for problem P2 makes use of a lower bound, LBND, on the optimal objective value which is used in step size determination using a variant of scheme (iii) given in Section II, as well as in the termination criterion each time the procedure is invoked. Again, we assume that both bounds are greater than zero.

ALGORITHM 3: UPPER BOUND ALGORITHM

```
1 Initialization
    Select $y \in S$.
    Initialize LBND and $\varepsilon$.
2 Find subgradient and step size
    Let $\bar{x}$ and $\Pi$ be the vectors of optimal primal and dual variables
      for Min $\{cx \mid Ax = b, \gamma \le x \le \theta\}$.
    UBND $\leftarrow c\bar{x}$.
    If (UBND - LBND) $\le \varepsilon$(UBND), terminate with $\bar{x}$ optimal;
      otherwise,
      $v_k \leftarrow -\Pi_{k_f} + \Pi_{k_t} + c_k,\ k = 1,2,\ldots,2K$.
      $\eta \leftarrow (v_1 + v_{K+1}, \ldots, v_K + v_{2K})$.
3 Move to new point
    (a) $y \leftarrow P[y - ((\mathrm{UBND} - \mathrm{LBND})/(2\|\eta\|^2))\eta]$.
    (b) Go to 2.
```

V. THE ALGORITHM

The solution of the equal flow problem using decomposition, as given in the previous section can be implemented without the lower bound procedure. It is also possible to implement the lower bound algorithm independently for the purpose of obtaining a lower bound on the optimal value of the equal flow problem. For the upper bound problem, some measure of the lower bound on the problem must be used to aid in termination. By merging the two procedures, an algorithm which adjusts the lower and upper bounds progressively can be used to advantage. Not only can such a procedure be used for obtaining feasible solutions with relative ease, but it can also provide a measure of how close this solution is to the optimal.

The algorithm for the solution of the equal flow problem iterates between the lower bound procedure and the upper bound procedure. The lower and upper bounds, LBND and UBND, progressively become tighter, closing in on the optimal solution to the problem. Each time the lower bound procedure is invoked, a maximum of ITERL iterations are performed. Each time the upper bound procedure is invoked, a maximum of ITERU iterations are performed. The tuning parameters for the algorithm are as follows: ITERL, ITERU, $\dot{p}$ (the initial step size), and ε (the termination criterion.)

ALGORITHM 4: RELAXATION/DECOMPOSITION ALGORITHM FOR THE EQUAL FLOW PROBLEM

```
0 Initialization
    Initialize ITERL, ITERU, $p$, $\varepsilon$.
    $T \leftarrow 0$, $R \leftarrow 0$, $w \leftarrow 0$, UBND $\leftarrow \infty$, LBND $\leftarrow -\infty$.
1 Compute Lower Bounds
    (a) Call ALGORITHM 2 (Steps 2 and 3 (a)).
    (b) $T \leftarrow T+1$
        If $T <$ ITERL, then go to step 1 (a).
2 Compute Upper Bounds
    (a) Call ALGORITHM 3 (Steps 2 and 3 (a)).
    (b) $R \leftarrow R+1$
        If $R <$ ITERU, then go to step 2 (a).
3 Reset iteration counts
    $T \leftarrow 0$, $R \leftarrow 0$, and go to step 1.
```

The initial equal flow allocation in the upper bound procedure makes use of the solution $\bar{x}$ to the last pure network flow problem solved in the lower bound procedure. The allocation for each of the K pairs of equal flow constraints is determined by:

$$y_k \leftarrow \min[\, u_k,\ (\bar{x}_k + \bar{x}_{K+k})/2 \,] \quad k = 1,2,\ldots,K.$$

All subsequent entries into Step 2 of the upper bound procedure use the most recent equal flow allocation in the previous upper bound iteration.

VI. COMPUTATIONAL RESULTS

The computer implementation of the algorithm is written in standard FORTRAN and not tailored to either the machine or FORTRAN compiler used for testing. The implementation, called EQFLO, makes use of MODFLO [1] to solve pure network subproblems. MODFLO is a set of subroutines which may be used to solve a network problem as well as reoptimize after problem data changes. Based on NETFLO [8], this code allows the user to change costs, bounds and/or requirements for a network problem and reoptimize.

The algorithm has been tested on a set of 10 test problems randomly generated using NETGEN [9], a large-scale network problem generator. The parameters used to generate test problems are described in Klingman, Napier, and Stutz [9]. The test problems have between 200 and 1500 nodes, and between 1500 and 6600 arcs. For each problem, the first 2K arcs were paired to form K equal flow side constraints. In order to gauge the performance of the algorithm for various values of K, some of the problems were generated using the same base network problem data with K varying from 75 to 200. The benchmark NETGEN problems have a specified percentage of arcs which are uncapacitated. For these arcs, the capacity was defined to be the maximum of all supplies and demands.

Computational testing was carried out on the IBM 3081D at Southern Methodist University using the FORTVS compiler with OPT = 2. In order to assess the computational gains afforded by the decomposition/relaxation algorithm for the equal flow problem, each problem was solved using MPSX [7]. An additional point of interest which was addressed is the choice of model for the equal flow problem when using a general linear programming system such as MPSX.

The equal flow problem can also be formulated as a network with side columns. For an equal flow problem defined on a network with m nodes, n arcs and K equal flow pairs, the side constraint formulation uses m + K constraints and n variables. The side column formulation has m constraints and n - K variables. It is best defined by partitioning the node-arc incidence matrix A =

[T:N] where T is $m \times 2K$ and N is $m \times n\text{-}2K$. The matrix T contains the first 2K columns of A, which correspond to arcs appearing in equal flow constraints. The side column formulation is given by

$$\begin{aligned} \text{minimize} \quad & fy + gz \\ \text{s.t.} \quad & Sy + Nz = r \\ & 0 \le z \le u \\ & 0 \le y \le U \end{aligned}$$

where, letting t_i, and s_i denote the ith columns of T and S,

$$\begin{aligned} s_k &= t_k + t_{K+k}, & k &= 1,2,\dots,K \\ f_k &= c_k + c_{K+k}, & k &= 1,2,\dots,K \\ U_k &= u_k, & k &= 1,2,\dots,K. \end{aligned}$$

Table I details the computational testing of the algorithm with parameters ITERL = 15, ITERU = 10 and p = .005. Of the 10 problems used, the first three are transportation problems (problems 5, 9, and 10), the next four are capacitated transshipment problems (problems 20, 21, 24, and 25) and the last three are uncapacitated transshipment problems (problems 28, 30, and 35). The test problems were formulated using both the side columns model and the side constraint model. Both models were solved using MPSX with default tuning parameters. The side column formulation ran slightly longer than the side constraint formulation on MPSX, even though it uses 75 fewer constraints and 75 fewer columns for the test problems in Table I.

For the test problems, EQFLO obtained feasible solutions whose objective function values were within 10% of the optimal in one-sixteenth of the time required by MPSX to obtain an optimum. For the equal flow problem with 400 constraints, 2692 arcs and 75 equal flow pairs, a 10% solution as well as a 5% solution are obtained in one-hundreth of the time required for solution by MPSX. For the 1500 node problem with 5880 arcs a 5% solution was obtained in 6% of the time required by MPSX. Guaranteed 5% solutions across all problems were obtained in about one-seventh of the MPSX time.

To determine the behavior of EQFLO as the number of side constraints increases and the stopping tolerance decreases, additional testing with problems 21, 24, and 28 was performed. Each of these base problems was used to generate equal flow problems with 75, 100, 150, and 200 equal flow constraints. The corresponding side column model was also generated. Table II summarizes the computational experience. The new algorithm performs favorably, when compared to MPSX, in obtaining guaranteed 5% solutions. For the 400 node problem with 2904 arcs and 100 equal flow contraints, a 1% solution is obtained in 3% of the time required by MPSX.

Table I. Comparison of EQFLO with MPSX (All Problems Have 75 equal flow pairs).

Problem Description			MPSX Time (in seconds) Formulation		EQFLO Time (in seconds)	
NETGEN Number	Nodes	Arcs	Side Constraints	Side Columns	ε 10%	5%
5	200	3100	11.4	12.0	1.6	11.9
9	300	6395	36.6	36.0	1.9	8.5
10	300	6611	33.6	33.6	4.4	19.2
20	400	1484	27.0	38.4	0.7	1.8
21	400	2904	67.8	69.0	0.8	0.8
24	400	1398	47.4	46.8	0.8	2.9
25	400	2692	112.8	105.0	1.0	1.0
28	1000	3000	45.6	51.6	2.2	2.8
30	1000	4500	45.6	69.0	14.8	16.9
35	1500	5880	102.0	144.0	4.6	5.8
			529.8	605.4	32.8	71.6

Table II. Effect of Decreasing ε and Increasing the Number of Equal Flow Pairs.

Problem Description		MPSX Time (in seconds) Formulation		EQFLO Time (in seconds)			
NETGEN Number	Number of Pairs	Side Constraints	Side Columns	ε 10%	5%	3%	1%
21	75	67.8	69.0	0.8	0.8	0.8	0.8
21	100	70.8	87.0	1.0	1.0	1.1	1.9
21	150	78.0	87.6	1.2	1.7	3.2	6.3
21	200	76.8	91.8	2.0	10.1	16.3	106.3
24	75	47.4	46.8	0.8	2.8	4.9	29.2
24	100	48.0	52.2	1.4	1.4	29.8	a.
24	150	76.2	51.0	11.7	61.0	144.5	a.
24	200	84.0	52.8	5.8	25.7	108.4	a.
28	75	45.6	51.6	2.2	2.8	8.9	12.0
28	100	42.0	70.1	3.1	3.9	14.4	107.5
28	150	46.2	87.0	4.2	15.2	46.3	a.
28	200	66.6	104.4	9.9	19.5	104.4	a.
		749.4	851.3	44.1	145.9	483.0	

a. Problem did not converge after 900 upper bound interations.

VII. SUMMARY AND CONCLUSIONS

The equal flow problem lends itself to solution by decomposition and relaxation. The use of these techniques in the solution procedure developed is advantageous because the essential solution mechanism required is the solution of sequences of pure network problems. By dispensing with the working basis required by other techniques, not only are computational efficiencies afforded but the natural characteristics of the problem enhanced.

The algorithm can be modified to assist in the solution of the integer equal flow problem. The lower bound automatically produces integer flows and the projection of the subgradient in the upper bound routine can be altered to require integrality on the equal flow allocation. Thus this solution procedure not only provides lower bounds on the integer equal flow problem efficiently, but it also has the inherent capability of producing feasible integer solutions with ease.

The development for the linear equal flow problem in this paper can be instructive in modelling and solving other network problems with specially structured side constraints such as proportional flow models used in manpower planning. The solution technique is best suited for a real-world situation in which one must quickly produce near optimal solutions.

REFERENCES

[1] Ali, A., E. Allen, R. Barr, and J. Kennington, "Reoptimization Procedures for Bounded Variable Primal Simplex Network Algorithms," *European Journal of Operational Research*, 23, 256-263 (1986).

[2] Allen E., R. Helgason, J. Kennington, and B. Shetty, "A Generalization of Polyak's Convergence Result for Subgradient Optimization," Technical Report 85-OR-7, Department of Operations Research, Southern Methodist University, Dallas, Texas, 75275 (1985), to appear in *Mathematical Programming*.

[3] Barr, R., K. Farhangian, and J. Kennington, "Networks with Side Constraints: An LU Factorization Update," *The Annals of the Society of Logistics Engineers.*, 1, 1, 66-85 (1986).

[4] Beck, P., L. Lasdon, and M. Engquist, "A Reduced Gradient Algorithm for Nonlinear Network Problems," *ACM Transactions on Mathematical Software*, 9, 57-70 (1983).

[5] Carraresi, P. and G. Gallo, "Network Models for Vehicle and Crew Scheduling," *European Journal of Operational Research*," 16, 139-151 (1984).

[6] Glover, F., R. Glover, and F. Martinson, "The U. S. Bureau of Land Management's New NETFORM Vegetation Allocation System," Technical Report of the Division of Information Science Research, University of Colorado, Boulder, Colorado, 80309 (1982).

[7] IBM Mathematical Programming System Extended/370 Program Reference Manual, File No. S370-82, IBM Corp., White Plains, New York (1979).

[8] Kennington, J., and R. Helgason, *Algorithms for Network Programming*, John Wiley and Sons, New York (1980).

[9] Klingman, D., A Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Minimum Cost Flow Network Problems," *Management Science*, 20, 814-821 (1974).

[10] Poljak, B. T., "A General Method of Solving Extremum Problems," *Soviet Mathematics Doklady*, 8, 3, 593-597 (1967).

[11] Shepardson, F., and R. Marsten, "A Lagrangean Relaxation Algorithm for the Two Duty Period Scheduling Problem," *Management Science*, 26, 274-281 (1980).

[12] Shetty, B., "The Equal Flow Problem," unpublished dissertation, Department of Operations Research, Southern Methodist University, Dallas, Texas, 75275 (1985).

[13] Shor, N., "On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems," Dissertation, Cybernetics Institute, Academy of Sciences, U.S.S.R. (1964).

[14] Turnquist, M., and C. Malandraki, "Estimating Driver Costs for Transit Operations Planning," Joint National Meeting of ORSA/TIMS, Dallas (1984).

[15] Zangwill, W., *Nonlinear Programming: A Unified Approach*, Prentice Hall, Englewood Cliffs, New Jersey (1969).

APPENDIX C

OPTIMAL TRUNK ASSIGNMENT SIMULATION USER'S GUIDE

Jeffery L. Kennington

Richard V. Helgason

Department of Operations Research
Southern Methodist University
Dallas, Texas 75275

John M. Colombi

John J. Salerno

RADC/DCLD
Griffiss AFB NY 13441-5700

I. INTRODUCTION

OTAS is a FORTRAN code that simulates the optimal assignment of voice and data traffic onto the trunks available to a node. Voice and data requests follow a Poisson arrival process and the service duration is exponential. Hence, at a given clock time one can have a voice request, a data request, or a disconnect from a previous request. A request is assigned on the least cost trunk which has the available capacity and delay requirements.

II. THE MODEL

The integer programming model describes mathematically the criteria for assigning a service request to an available trunk. If no trunk is available, the model has no solution and the service is denied in the simulation.

2.1 Subscript

Let i - denote a trunk

2.2 Constants

Let $\bar{d}$ - denotes the maximum allowable delay of the requested service (micro-secs).

$\bar{c}$ - denotes the required capacity of the requested service (K bits/sec).

d_i - denotes the delay of the ith trunk (micro-secs).

c_i - denotes the capacity of the ith trunk (K bits/sec).

a_i - denotes the availablity of the ith trunk (%).

u_i - denotes the current usage of the ith trunk (K bits/sec).

v_i - denotes the unit cost for the ith trunk (\$ (K bit/sec)).

2.3 Decision Variables

Let x_i - be 1 if the service is assigned to trunk i and, 0 otherwise.

2.4 Constraints

(TRUNK CAPACITY)

$\bar{c}x_i \leq c_i a_i - u_i$: for all i.

(DELAY)

$d_i x_i \leq \bar{d}$; for all i

(SELECT ONE)

$\sum_i x_i = 1$

(INTEGRALITY)

$x_i = 0, 1$; for all i

(OBJECTIVE FUNCTION)

minimize $\sum_i v_i x_i$

III. INPUT

The input consist of three files corresponding to the service data, the trunk data, and a control file to manage the simulation length and output.

3.1 Tape 1: Service Input Data

(All data are in free format one entry/line. Please omit decimal points for integer data.)

Cols	Type	Name	Description
1-80	Integer	INTV	Inter-arrival time between calls (secs) (Assume exponential distribution)
1-80	Integer	SERV	Expected call duration (secs) (Assume exponential distribution)
1-80	Integer	INTD	Inter-arrival time between data requests (secs)
1-80	Integer	SERD	Expected service duration (secs) (Assume exponential distribution)
1-80	Real	CAPV	Service capacity required for voice (K bits/sec)
1-80	Real	CAPD	Service capacity required for data (K bits/sec)
1-80	Integer	DELV	Maximum allowable delay for voice service (micro-secs)
1-80	Integer	DELD	Maximum allowable delay for data service (micro-secs)

3.2 Tape 2: Trunk Input Data

(Name must appear in columns 1 through 8. All other data are in free format one entry/line. Please omit decimal points for integer data. Trunk 2 follows trunk 1, etc.)

Cols	Type	Name	Description

1-80	Character	NAME(I)	Trunk I name (8 characters)
			Examples include:
			LL = Land Line
			SAT = Satellite
			HF = High Frequency
			LOS = Line of Sight

1-80	Real	CAPAC(I)	Trunk I capacity (K bits/sec)
1-80	Integer	DELAY(I)	Trunk I delay time (micro-secs)
1-80	Integer	AVAIL(I)	Trunk I availablity (0% - 100%)
1-80	Real	COST(I)	Trunk I unit cost ($/(K bits/sec))

3.3 Tape 3: Simulation Control Data

(All data are in free format one entry/line. Please omit decimal points for integer data.)

Cols	Type	Name	Description
1-80	Integer	TOTALT	Total time of simulation in seconds
1-80	Integer	PRINT	Level of intermediate output 0 - No output 1 - Print status after 1% of total time 2 - Print every event

```
FILE: SERVICE  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

        60       SECS             (60 CALL/HOUR)
       240       SECS             (4 MINUTES/CALL)
        60       SECS             (60 DATA REQUESTS/HOUR)
       300       SECS             (5 MINUTES/REQUEST)
        64.      K BITS/SEC       (VOICE CAPACITY REQUIRED)
         9.6     K BITS/SEC       (DATA  CAPACITY REQUIRED)
       250       MICRO-SECS       (MAX VOICE DELAY ALLOWABLE)
      1000       MICRO-SECS       (MAX DATA  DELAY ALLOWABLE)
```

```
FILE: TRUNK    DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

LL              (LAND LINE)            TYPE
  192           (K BITS/SEC)           CAPACITY
  100           (MICRO-SECS)           DELAY
  100           (%)                    AVAILABILITY
     .0015      ( $/K BITS/SEC) )      COST
SAT             (SATELLITE)            TYPE
  128           (K BITS/SEC)           CAPACITY
  500           (MICRO-SECS)
   50           (%)                    AVAILABILITY
     .0007      ( $/K BITS/SEC) )      COST
HF              (HIGH FREQUENCY)       TYPE
   48.          (K BITS/SEC)           CAPACITY
  250           (MICRO-SECS)           DELAY
  100           (%)                    AVAILABILITY
     .0003      ( $/K BITS/SEC) )      COST
LOS             (LINE OF SIGHT)        TYPE
  128           (K BITS/SEC)           CAPACITY
  150           (MICRO-SECS)
   85           (%)                    AVAILABILITY
     .0005      ( $/K BITS/SEC) )      COST
```

```
FILE: CONTROL  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

 7200     (SECS)          20 MINUTE RUN
    0                     PRINT LEVEL
```

IV. OUTPUT REPORTS

Three reports are generated by the simulation on files 7, 8, and 9. Reports 1 and 2 give a summary of the statistics kept on voice and data service. Report 3 gives a summary of the activity on each trunk. Sample reports follow:

```
FILE: REPORT1  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

1REPORT 1:  SERVICE STATISTICS FOR VOICE REQUESTS

 INPUT:     EXPECTED INTER-ARRIVAL TIME              60 (SECS)
 INPUT:     EXPECTED DURATION                       240 (SECS)
 INPUT:     REQUIRED CAPACITY                   64.0000 (K BITS/SEC)
 INPUT:     MAXIMUM ALLOWABLE DELAY                 250 (MICRO-SECS)
 OUTPUT:    TOTAL REQUESTS                          121
 OUTPUT:    NUMBER OF BLOCKED REQUESTS               41
 OUTPUT:    % REQUESTS BLOCKED                       33 (%)
 OUTPUT:    TOTAL SIMULATION TIME                  7239 (SECS)
```

```
FILE: REPORT2  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

1REPORT 2:  SERVICE STATISTICS FOR DATA REQUESTS

 INPUT:     EXPECTED INTER-ARRIVAL TIME              60 (SECS)
 INPUT:     EXPECTED DURATION                       300 (SECS)
 INPUT:     REQUIRED CAPACITY                    9.6000 (K BITS/SEC)
 INPUT:     MAXIMUM ALLOWABLE DELAY                1000 (MICRO-SECS)
 OUTPUT:    TOTAL REQUESTS                          124
 OUTPUT:    NUMBER OF BLOCKED REQUESTS                0
 OUTPUT:    % REQUESTS BLOCKED                        0 (%)
 OUTPUT:    TOTAL SIMULATION TIME                  7239 (SECS)
```

```
FILE: REPORT3  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

1REPORT 3:  TRUNK USAGE

 TRUNK NUMBER:        1

 NAME                LL

 CAPACITY         192.00 (K BITS/SEC)

 DELAY               100 (MICRO-SECS)

 AVAILABILITY        100 (%)

 UNIT COST       0.00150 (  $/(K BITS/SEC)  )

 T R U N K    U T I L I Z A T I O N
 - - - - -    - - - - - - - - - - -

 UTILIZATION      TIME (SECS)     % TOTAL TIME
 -----------      ---- ------     - ----- ----

    0- 10 %            846         11.69
   11- 20 %              0          0.00
   21- 30 %              0          0.00
   31- 40 %           1274         17.60
   41- 50 %              0          0.00
   51- 60 %              0          0.00
   61- 70 %           2422         33.46
   71- 80 %              0          0.00
   81- 90 %              0          0.00
   91-100 %           2697         37.26

   TOTAL              7239
1REPORT 3:  TRUNK USAGE

 TRUNK NUMBER:        2

 NAME                SAT

 CAPACITY         128.00 (K BITS/SEC)

 DELAY               500 (MICRO-SECS)

 AVAILABILITY         50 (%)

 UNIT COST       0.00070 (  $/(K BITS/SEC)  )

 T R U N K    U T I L I Z A T I O N
 - - - - -    - - - - - - - - - - -
```

```
FILE: REPORT3  DATA     A1  SOUTHERN METHODIST UNIVERSITY -- CMS RELEASE 4.0

 UTILIZATION      TIME (SECS)      % TOTAL TIME
 -----------      ---- ------      - ----- ----

   0- 10 %           6547            90.44
  11- 20 %            152             2.10
  21- 30 %            474             6.55
  31- 40 %             66             0.91
  41- 50 %              0             0.00
  51- 60 %              0             0.00
  61- 70 %              0             0.00
  71- 80 %              0             0.00
  81- 90 %              0             0.00
  91-100 %              0             0.00

   TOTAL             7239
1REPORT 3:  TRUNK USAGE

 TRUNK NUMBER:          3

 NAME                  HF

 CAPACITY            48.00 (K BITS/SEC)

 DELAY                 250 (MICRO-SECS)

 AVAILABILITY          100 (%)

 UNIT COST         0.00030 (  $/(K BITS/SEC)  )

 T R U N K     U T I L I Z A T I O N
 - - - - -     - - - - - - - - - - -

 UTILIZATION      TIME (SECS)      % TOTAL TIME
 -----------      ---- ------      - ----- ----

   0- 10 %            144             1.99
  11- 20 %            607             8.39
  21- 30 %              0             0.00
  31- 40 %           1591            21.98
  41- 50 %              0             0.00
  51- 60 %           1128            15.58
  61- 70 %              0             0.00
  71- 80 %           1709            23.61
  81- 90 %              0             0.00
  91-100 %           2060            28.46

   TOTAL             7239
1REPORT 3:  TRUNK USAGE
```

```
TRUNK NUMBER:        4

NAME                LOS

CAPACITY         128.00 (K BITS/SEC)

DELAY               150 (MICRO-SECS)

AVAILABILITY         85 (%)

UNIT COST       0.00050 (  $/(K BITS/SEC)  )

T R U N K    U T I L I Z A T I O N
- - - - -    - - - - - - - - - - -

UTILIZATION      TIME (SECS)     % TOTAL TIME
-----------      ---- ------     - ----- ----

  0- 10 %             613            8.47
 11- 20 %             147            2.03
 21- 30 %             215            2.97
 31- 40 %               0            0.00
 41- 50 %            2478           34.23
 51- 60 %            1170           16.16
 61- 70 %            1523           21.04
 71- 80 %            1093           15.10
 81- 90 %               0            0.00
 91-100 %               0            0.00

 TOTAL               7239
```

APPENDIX D

A PARALLELIZATION OF THE SIMPLEX METHOD

R. V. Helgason

J. L. Kennington

Department of Operations Research
Southern Methodist University
Dallas, Texas 75275

H. A. Zaki

Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champlaign
Urbana, Illinois 61801

ACKNOWLEDGEMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFOSR 83-0278, the Department of Defense under Contract Number MDA 903-86-C-0182, and Rome Air Development Center under Contract Number SCEEE PDP/86-75.

ABSTRACT

This paper presents a parallelization of the simplex method for linear programming. Current implementations of the simplex method on sequential computers are based on a triangular factorization of the inverse of the current basis. An alternative decompostion designed for parallel computation, called the quadrant interlocking factorization, has previously been proposed for solving linear systems of equations. This research presents the theoretical justification and algorithms required to implement this new factorization in a simplex-based linear programming system.

I. INTRODUCTION

The introduction of parallel computers into scientific computing in the past decade is the beginning of a new era. The invention of new algorithms will be required to ensure realization of the potential of these and future architectural improvements in computers. Already the use of parallel computers has given rise to studies in concurrency factors, vectorization, and asynchronous procedures. These have led to multifold increases in speed over conventional serial machines after the calculations have been rearranged to take advantage of the specific hardware. This paper presents a parallelization of the simplex algorithm for general linear programs. Our work begins with new results for solving systems of linear equations and is directed toward the hardware design currently adapted by Sequent Computer Systems, Inc. of Beaverton, Oregon.

The following notation is used throughout this paper. Let $B_{i:j,k:l}$ represent a submatrix of B composed of rows i through j and columns k through l. If $i=j$ $(k=l)$, we write $B_{i,k:l}$ $(B_{i:j,k})$. The j^{th} row (column) of B is denoted by $B_{j,.}$ $(B_{.,j})$. The i,j^{th} element of B is $B_{i,j}$.

The linear programming problem is represented mathematically as follows:

$$\begin{aligned} &\text{minimize} \quad c^T x \\ &\text{subject to } \; Ax = b \\ &\qquad\qquad 0 \le x \le u, \end{aligned}$$

where A is a known m by n matrix, all other quantities are conformable, and all vectors are known except x.

The upper bounded version of Dantzig's simplex method for solving the linear programming problem may be stated as follows:

<u>*Algorithm 1.1 : The Simplex Method*</u>

0. Initialization

Let $[x^B \mid x^N]$ be a basic feasible solution with $A = [B \mid N]$. Let the cost vector $[c^B \mid c^N]$ and bounds $[u^B \mid u^N]$ be partitioned similarly. Assume that B^{-1} is available in some factored form. Initialize *iter* to 0 and the reinversion frequency, *freq*.

1. Calculate the Dual Variables (BTRAN)

$$\pi \leftarrow c^B B^{-1}. \tag{1.1}$$

2. Pricing

Let $K_1 = \{j : x_j^N = 0 \text{ and } c_j^N - \pi N_{.,j} < 0\}$,

and $K_2 = \{j : x_j^N = u_j^N \text{ and } c_j^N - \pi N_{.,j} > 0\}$.

If $K_1 \cup K_2 = \Phi$, terminate with $[x^B \mid x^N]$ optimal;

otherwise, select $k \varepsilon K_1 \cup K_2$ and set

$$\delta \leftarrow \begin{cases} 1, \text{ if } k \varepsilon K_1 \\ -1, \textit{otherwise.} \end{cases}$$

3. Column Update (FTRAN)

$$y \leftarrow B^{-1} N_{.,k} \tag{1.2}$$

4. Ratio Test

$$\Delta_1 \leftarrow \min_{sign(y_j) = sign(\delta)} \left\{ \frac{x_j^B}{|y_j|}, \infty \right\}$$

$$\Delta_2 \leftarrow \min_{sign(y_j) = sign(-\delta)} \left\{ \frac{u_j^B - x_j^B}{|y_j|}, \infty \right\}$$

$$\Delta \leftarrow \min \left\{ \Delta_1, \Delta_2, u_k^N \right\}.$$

5. Right Hand Side Update

$$x_k^N \leftarrow x_k^N + \Delta\delta$$

$$x^B \leftarrow x^B - \Delta\delta y.$$

If $\Delta = u_k^N$, return to 1.

6. Basis Inverse Update

Let p denote the index of x^B which produced Δ and set

$$\eta \leftarrow \begin{cases} -y_i / y_p \text{ , if } i \neq p \\ 1/y_p \text{ ; } \textit{otherwise}, \end{cases}$$

$$E \leftarrow I - e_p e_p^T + \eta e_p^T$$

$$\bar{B}^{-1} \leftarrow EB^{-1}. \tag{1.3}$$

7. Reinversion Check

iter $\leftarrow$ *iter* + 1.

If $mod(iter, freq) = 0$, then refactor $\bar{B}^{-1}$.

Return to 1 using $\bar{B}^{-1}$ as B^{-1}, the current basis inverse.

Two of the most common factorizations of the basis matrix inverse are the product form and the elimination form, which correspond to the methods for solution of linear equations known as Gauss-Jordan reduction and Gauss reduction (LU factorization), respectively, where L is a lower triangular matrix and U is an upper triangular matrix. The elimination form produces a sparser representation of the basis inverse than the product form, and accordingly leads to faster implementation of a simplex iteration and a considerable savings in storage.

Historically, the elimination form of the inverse, due to Markowitz [1957-1], was the first LU factorization method and was introduced to preserve sparsity during reinversion. However, once reinversion was completed further pivot operations were handled using product form. Bartels and Golub proposed updating L and U in a numerically stable way. (see Bartels [1971-1]). Their updating scheme tends to promote the growth of nonzeros in U, leading to a potentially severe loss of sparsity. Forrest and Tomlin [1972-1] designed a different updating scheme for the triangular factors to preserve sparsity at some sacrifice in numerical stability. Subsequent implementation of the Bartels-Golub method, designed by Reid [1982-1] and Saunders [1976-1], combine the virtues of accuracy and speed.

Several parallel versions of the LU factorization algorithm for solving general linear systems of equations are presently available (Chen et al. [1984-1] and Dongarra and Sorensen [1984-2]). All versions are based on restructuring the original serial algorithm to reveal possible independent tasks that can be carried out concurrently.

Evans and Hatzopoulos [1979-1] proposed a matrix factorization, called the Quadrant Interlocking Factorization (QIF), as an appropriate tool for solving linear systems on parallel computers. The QIF is similar to the LU factorization, but is claimed to be more suitable for concurrent computation.

This paper presents a parallelization of the simplex method using the QIF. The outline of the paper is as follows. In Section II, the QIF is developed. An algorithm for updating the QIF of B^{-1} is presented in Section III. Mathematically, the problem is to efficiently obtain a factorization of $\bar{B}^{-1}$ (see step 6 of Algorithm 1.1) from the factorization of B^{-1}. In Section IV, we develop a parallelization of the reinversion routine used in step 7 and propose a parallel implementation of both the BTRAN and FTRAN operations of steps 1 and 3.

The parallel algorithms presented in this study are designed for a MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of applying all their power to a single job in a timely and coordinated manner. The Balance Systems 8000 and 21000 from Sequent Computer Systems are examples of such machines.

II. THE QUADRANT INTERLOCKING FACTORIZATION

In this section we describe a matrix factorization suggested by Evans and Hatzopoulos [1979-1] known as *the Quadrant Interlocking Factorization (QIF)*. This decomposition is designed to solve linear systems on parallel computers (see Evans and Hatzopoulos [1979-1], Evans and Hadjidimos [1980-1], Evans [1982-1] and Feilmeier [1982-1]). The factors and some of their characteristics are described in Section 2.1. We show that any nonsingular matrix can be factorized into its QIF in two ways, the *Forward QIF* and the *Backward QIF* The factorization algorithms are developed in Sections 2.2 and 2.3. The relationship of quadrant and triangular matrices is presented in Section 2.4.

2.1 The Quadrant Interlocking Factors

Consider the following matrix

$$W = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ w_{2,1} & 1 & \cdots & 0 & w_{2,m} \\ w_{3,1} & w_{3,2} & \cdots & w_{3,m-1} & w_{3,m} \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ w_{m-2,1} & w_{m-2,2} & \cdots & w_{m-2,m-1} & w_{m-2,m} \\ w_{m-1,1} & 0 & \cdots & 1 & w_{m-1,m} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} . \tag{2.1}$$

Note that the non-arbitrary entries of W are given by

$$w_{i,j} = \begin{cases} 1, \; i=j; \\ 0, \; i=1,\ldots,[m/2] \; , \; j=(i+1),\ldots,(m-i+1); \\ 0, \; i=\overline{m},\ldots,m \; , \; j=m-i+1,\ldots,i-1; \end{cases} \tag{2.2}$$

where

$[x]$ = the largest integer not greater than the value of x
$\overline{m} = m + 1 - [m/2]$.

Also, consider the matrix

$$Z = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,m-1} & z_{1,m} \\ 0 & z_{2,2} & \cdots & z_{2,m-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & z_{m-1,2} & \cdots & z_{m-1,m-1} & 0 \\ z_{m,1} & z_{m,2} & \cdots & z_{m,m-1} & z_{m,m} \end{bmatrix} . \tag{2.3}$$

Note that

$$z_{i,j} = 0, \begin{cases} j=1,\ldots,[(m-1)/2]\ ,\ i=j+1,\ldots,m-j\,; \\ j=[m/2]+2,\ldots,m\ \ ,\ i=m+2-j,\ldots,j-1. \end{cases} \tag{2.4}$$

Any square matrix may be partitioned by its diagonal and secondary diagonal into four quadrants. The potentially nonzero elements of W are in the left and right quadrants while those of Z are in the upper and lower quadrants. Therefore, we call any square matrix whose nonzero structure follows (2.1) and (2.2), or one that can be brought to such a form by row and/or column interchanges a *left-right quadrant (LRQ)* matrix. Similarly, any square matrix whose nonzero structure follows (2.3) and (2.4), or one that can be brought to such a form by row and/or column interchanges is called an *upper-lower quadrant (ULQ)* matrix . Examples of W and Z matrices for an odd and an even m are given below:

Example 2.1 (m=5)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & w_{2,5} \\ w_{3,1} & w_{3,2} & 1 & w_{3,4} & w_{3,5} \\ w_{4,1} & 0 & 0 & 1 & w_{4,5} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & 0 \\ 0 & 0 & z_{3,3} & 0 & 0 \\ 0 & z_{4,2} & z_{4,3} & z_{4,4} & 0 \\ z_{5,1} & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} \end{bmatrix}.$$

<u>Example 2.2</u> (m=6)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & 0 & w_{2,6} \\ w_{3,1} & w_{3,2} & 1 & 0 & w_{3,5} & w_{3,6} \\ w_{4,1} & w_{4,2} & 0 & 1 & w_{4,5} & w_{4,6} \\ w_{5,1} & 0 & 0 & 0 & 1 & w_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\ 0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\ 0 & 0 & z_{4,3} & z_{4,4} & 0 & 0 \\ 0 & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\ z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6} \end{bmatrix}.$$

Without loss of generality we assume that m is even. For linear programming, we can always append a nonbinding constraint so that the total number of constraints is even.

The set of all LRQ matrices of order m is denoted by $\{M_m^w\}$ and the set of all ULQ matrices of order m is denoted by $\{M_m^z\}$. Let A $\varepsilon R^{m,m}$ and $\bar{A}=A_{i,j}.e_i.e_j^T$. If $(\bar{A}+I)\varepsilon\{M_m^w\}$ we say that $A_{i,j}$ is a *W-element* ; otherwise, it is a *non-W-element* . Similarly, if $\bar{A}\,\varepsilon\{M_m^z\}$ we say that $A_{i,j}$ is a *Z-element* ; otherwise, it is a *non-Z-element.*

<u>Proposition 2.1</u>

$\{M_m^w\}$ and $\{M_m^z\}$ are closed under addition, scalar multiplication, multiplication and inversion .

(The proof of this Proposition may be found in Zaki [1986-1]).

2.2 The Forward Quadrant Interlocking Factorization Algorithm

In this section we present an algorithm which obtains the WZ factorization of any nonsingular matrix. That is, given a nonsingular matrix B, find W and Z such that $B = WZQ$, where Q is a permutation matrix. This factorization is analogous to the LU

factorization in common use in many production linear programming packages.

Definition 2.1

An elementary left-right quadrant (ELRQ) matrix of order m and index k is a matrix of the form:

$$N^k = I - u^k \cdot e_k^T - v^k \cdot e_l^T \tag{2.5}$$

where

$$l = m - k + 1 \qquad , k \; \varepsilon 1,2,\ldots,(m/2)-1, \tag{2.6}$$
$$e_i^T \cdot u^k = 0 \quad \text{and} \quad e_i^T \cdot v^k = 0 \quad \text{for } i = 1,2,\ldots,k, l, l+1,\ldots,m. \tag{2.7}$$

The conditions (2.7) require that the first k and last k components of u^k and v^k be zero, that is, u^k and v^k have the form:

$$u^k = (0,0,\ldots,0,u_{k+1}^k, u_{k+2}^k, \ldots, u_{m-k}^k, 0,0,\ldots,0)^T \tag{2.8}$$
$$v^k = (0,0,\ldots,0,v_{k+1}^k, v_{k+2}^k, \ldots, v_{m-k}^k, 0,0,\ldots,0)^T . \tag{2.9}$$

In general an ELRQ matrix of order m and index k has the form depicted in Figure 2.1. Thus, an ELRQ matrix of index k is a LRQ matrix whose only nonidentity columns are columns k and l ($l = m - k + 1$). ELRQ matrices are easily inverted. It is apparent that

$$\left[N^k\right]^{-1} = I + u^k \cdot e_k^T + v^k \cdot e_l^T \tag{2.10}$$

which is also an ELRQ matrix of index k.

Proposition 2.2

Let

$$N^{(k)} = N^1 N^2 \cdots N^k \tag{2.11}$$

where N^i is an ELRQ matrix of index i , i=1,2,...,k. Then $N^{(k)}$ is a LRQ matrix whose j^{th} and $(m-j+1)^{st}$ columns are those of N^j.

(The proof of this Proposition may be found in Zaki [1986-1]).

Definition 2.2

A partially reduced upper-lower quadrant (PRULQ) matrix of index k and order m is a square matrix whose non-Z elements are zero in columns 1 through $k-1$ and $l+1$ through m, where $k = 1,2,\ldots,m/2$ and $l = m-k+1$. Its general form is shown in Figure 2.2. Note that B^1 has no special zero structure and $B^{m/2}$ is an ULQ matrix.

Proposition 2.3

Let B^k be a PRULQ matrix of index k. If B^k is nonsingular then there exist j_1 and j_2 such that $k \le j_1 < j_2 \le l$ and

$$\delta = B^k_{k,j_1} \cdot B^k_{l,j_2} - B^k_{k,j_2} \cdot B^k_{l,j_1} \neq 0. \qquad (2.12)$$

Proof

Suppose $\delta = 0$ for every $k \le j_1 < j_2 \le l$. Then $B^k_{k,.}$ must be a multiple of $B^k_{l,.}$. This contradicts the assumption that B^k is nonsingular.

Permuting the columns of a PRULQ matrix so that certain elements provide a nonsingular 2x2 submatrix is analogous to interchanging rows and columns in matrix inversion to obtain a nonzero pivot element. Now, let B^k be a nonsingular PRULQ matrix of index k. Let j_1 and j_2 satisfy Proposition 2.3 and define Q^k to be the permutation matrix such that

$$\bar{B}^k = B^k Q^k$$

where

$$\bar{B}^k_{.,k} = B^k_{.,j_1} \quad \text{and} \quad \bar{B}^k_{.,l} = B^k_{.,j_2} . \qquad (2.13)$$

Let A be any square matrix of order m and let $k \varepsilon \{1,\ldots,m/2\}$. Define $S^k(A)$ to be the following 2x2 principal submatrix of A

$$S^k(A) = \begin{bmatrix} A_{k,k} & A_{k,l} \\ A_{l,k} & A_{l,l} \end{bmatrix} \qquad (2.14)$$

where $l = m-k+1$. Using these definitions and Proposition 2.3, it is clear that $\bar{B}^k = B^k Q^k$ is a nonsingular PRULQ matrix of index k and $S^k(\bar{B}^k)$ is nonsingular.

We now show how one may transform a PRULQ matrix of index k into a PRULQ

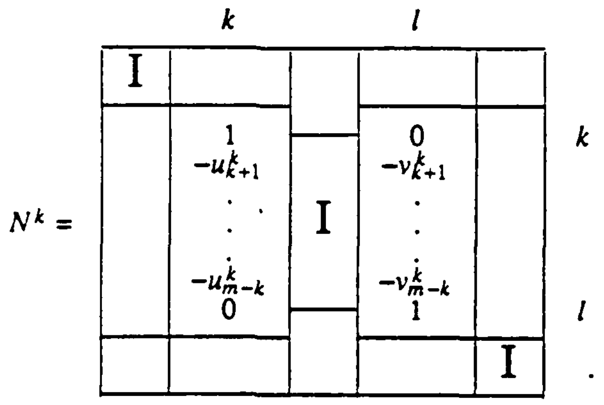


Figure 2.1. Illustration of the ELRQ matrix of order m and index k.

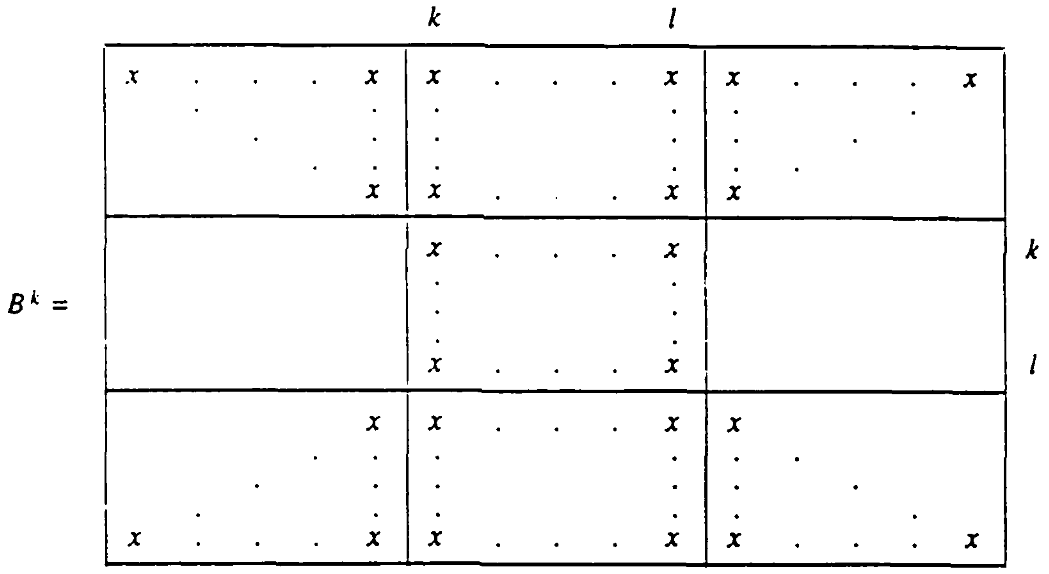


Figure 2.2. Illustration of a PRULQ matrix of order m and index k.

matrix of index $k+1$.

Proposition 2.4

Let B^k be a nonsingular PRULQ matrix of index k and let Q^k be the permutation matrix that interchanges columns k and $m-k+1$ with columns j_1 and j_2, respectively, where j_1 and j_2 are obtained so that they satisfy Proposition 2.3. Let N^k be an ELRQ matrix of index k whose u^k and v^k vectors are determined by solving the following $(m-2k)$ 2x2 linear systems

$$\begin{bmatrix} u_i^k & v_i^k \end{bmatrix} \; S^k(\bar{B}^k) = \begin{bmatrix} \bar{B}_{i,k}^k & \bar{B}_{i,l}^k \end{bmatrix} \qquad , i=k+1,\ldots,m-k. \tag{2.15}$$

Then $B^{k+1} = N^k \, B^k \, Q^k$ is a nonsingular PRULQ matrix of index $k+1$.

Proof

Since $\bar{B}^k$ is nonsingular and N^k is nonsingular, then B^{k+1} is nonsingular. B^{k+1} is a PRULQ matrix of index $k+1$ if all non-Z-elements in columns 1 through k and l through m are zero. Since $\bar{B}^k$ is a PRULQ matrix of index k , we only need to show that the effect of N^k on $\bar{B}^k$ is to zero out the non-Z-elements in columns k,l. To show this, we begin by rewriting (2.15) as

$$\begin{bmatrix} u_i^k & v_i^k \end{bmatrix} \begin{bmatrix} \bar{B}_{k,k}^k & \bar{B}_{l,k}^k \\ \bar{B}_{k,l}^k & \bar{B}_{l,l}^k \end{bmatrix} = \begin{bmatrix} \bar{B}_{i,k}^k & \bar{B}_{i,l}^k \end{bmatrix}$$

or for $i = k+1, k+2, \ldots, m-k$

$$u_i^k \cdot \bar{B}_{k,k}^k + v_i^k \cdot \bar{B}_{l,k}^k = \bar{B}_{i,k}^k \tag{2.16}$$
$$u_i^k \cdot \bar{B}_{k,l}^k + v_i^k \cdot \bar{B}_{l,l}^k = \bar{B}_{i,l}^k. \tag{2.17}$$

We now consider the non-Z-elements of $B_{.,k}^{k+1}$.

For $i = k+1, k+2 \ldots, m-k$

$$B_{i,k}^{k+1} = N_{i,.}^k \cdot \bar{B}_{.,k}^k$$
$$= -u_i^k \cdot \bar{B}_{k,k}^k - v_i^k \cdot \bar{B}_{l,k}^k + \bar{B}_{i,k}^k = 0 \quad by\ (2.16). \tag{2.18}$$
$$B_{i,l}^{k+1} = N_{i,.}^k \cdot \bar{B}_{.,l}^k$$
$$= -u_i^k \cdot \bar{B}_{k,l}^k - v_i^k \cdot \bar{B}_{l,l}^k + \bar{B}_{i,l}^k = 0 \quad by\ (2.17). \tag{2.19}$$

$$B_{i,j}^{k+1} = N_{i,\cdot} \bar{B}_{\cdot,j}^{k} = 0 \quad \text{for } j=1,\dots,k-1 \text{ and } l+1,\dots,m. \tag{2.20}$$

Also we note that the desired zeros created in earlier stages in $\bar{B}^k$ are not affected by N^k, since for $i=1,\dots,k-1,l+1,\dots,m$

$$B_{i,\cdot}^{k+1} = N_{i,\cdot}^{k} \cdot \bar{B}^k = e_i \cdot \bar{B}^k = \bar{B}_{i,\cdot}^{k}. \tag{2.21}$$

From (2.18) through (2.21) we conclude that B^{k+1} is a PRULQ matrix of index $k+1$.

Given the above definitions, the forward quadrant interlocking factorization algorithm may be stated as follows.

Algorithm 2.1 : The Forward Quadrant Interlocking Factorization

Let $B \varepsilon R^{m,m}$. The following steps decompose B to its quadrant interlocking factors with $B = W Z Q$.

```
Initialize
        $B^1 = B$,
        $K = m/2$.
Main Loop
  For $k = 1,2,\dots,K-1$
      1. Column Permutation
          Find $j_1$ and $j_2$ satisfying Proposition 2.3.
          If none exists, then terminate with the conclusion that $B$ is singular.
          Otherwise, construct $Q^k$ using $j_1$ and $j_2$.
      2. Compute the vectors $u^k$, $v^k$
          by solving the $(m-2k)$ 2x2 linear systems, (2.15).
      3. Construct $N^k$
                  $N^k = I - u^k \cdot e_k^T - v^k \cdot e_l^T$.
      4. Construct $B^{k+1}$
                  $B^{k+1} = N^k\ B^k\ Q^k$.
  Next $k$
```

Proposition 2.5

Let B be a nonsingular matrix of size m. Then *Algorithm 2.1* decomposes B to its forward quadrant interlocking factors ,

$$B = W\ Z\ Q \tag{2.22}$$

where

(1) $W \varepsilon \{M_m^w\}$, $W = (N^{K-1}N^{K-2} \cdots N^1)^{-1}$,
(2) $Z \varepsilon \{M_m^z\}$, $Z = B^K$,and
(3) Q is a permutation matrix , $Q = (Q^1 Q^2 \cdots Q^{K-1})^{-1}$.

Proof

Let $B^1 = B$. Applying Proposition 2.4 for k =1,2,...,$(m/2)-1$, we obtain

$$B^K = N^{K-1}\, N^{K-2} \cdots N^1 B^1 Q^1 \cdots Q^{K-2} Q^{K-1}, \tag{2.23}$$

where B^K is an ULQ matrix, N^j, $j = 1,...,K-1$ are ELRQ matrices as computed in (2.15) and Q^j are permutation matrices. From (2.23),

$$B^1 = (N^{K-1}\, N^{K-2} \cdots N^1)^{-1} B^K\ (Q^1 \cdots Q^{K-2} Q^{K-1})^{-1}. \tag{2.24}$$

Let $N^{(K-1)} = (N^{K-1}\, N^{K-2} \cdots N^1)^{-1}$. By Proposition 2.2 $N^{(K-1)}$ is a LRQ matrix. Also, let $Q^{(K-1)} = (Q^1 \cdots Q^{K-2} Q^{K-1})^{-1}$. Since the product of permutation matrices is a permutation matrix, $Q^{(K-1)}$ is a permutation matrix. Thus, (2.24) can be written as

$$B^1 = B = N^{(K-1)}\ B^K\ \ Q^{(K-1)}, \tag{2.25}$$

and (2.22) follows by setting $W = N^{(K-1)}$, $Z = B^K$, and $Q = Q^{(K-1)}$ in (2.25).

Proposition 2.6

Algorithm 2.1 without column permutations requires

$$m^3/3 + m^2/2 - 4m/3$$

multiplications on a sequential machine.

Proof

Ignoring column permutations, we trace the operations in the main loop excluding step 1

The number of multiplications to compute u^k and v^k

$$= \sum_{k=1}^{K-1} [\, 2 + 6(m-2k) \,]$$

$$= m + 3m\,(m-2)/2. \tag{2.26}$$

The number of multiplications to compute B^{k+1}

$$= 2. \sum_{k=1}^{K-1} (m-2k)^2$$

$$= m.\,(m-1).(m-2)/3. \tag{2.27}$$

Summing (2.26) and (2.27) we obtain the specified total number of multiplications.

In Algorithm 2.1 the columns of the PRULQ matrix are permuted to find a 2x2 matrix with a nonzero determinant. There are obvious alternatives that may be used. To ensure numerical stability for instance, we may find the matrix whose determinant has the largest absolute value, or the matrix that has the smallest condition number. Another approach is to permute the rows of the PRULQ matrix to find the required nonsingular 2x2 matrix attempting to minimize fill-in in the nonpivot rows. Both row and/or column permutations can be selected on numerical stability and/or sparsity grounds.

2.3 The Backward Quadrant Interlocking Factorization Algorithm

Unlike the triangular factors (L,U) of a matrix, the quadrant interlocking factors (W,Z) possess different potential density. That is, the number of potentially nonzero elements in W is different than that in Z. In this section we present an algorithm which obtains the *ZW* factorization of any nonsingular matrix. We refer to this algorithm as the *Backward* QIF algorithm, as opposed to the *Forward* QIF algorithm of Section 2.2 that produces the *WZ* factorization. The development of this algorithm is very similar to the previous one. The proofs of Propositions 2.7 through 2.10 in this section, use arguments similar to those used in Propositions 2.2 through 2.5 and hence are omitted.

Definition 2.3

An elementary upper lower quadrant (EULQ) matrix of order m and index k is a matrix of the form :

$$M^k = I_m - r^k \cdot e_k^T - s^k \cdot e_l^T - e_k \cdot e_k^T - e_l \cdot e_l^T \tag{2.28}$$

where

$$\begin{aligned} &l = m - k + 1, k \varepsilon 1,2, \cdots, m/2, \\ &e_i^T \cdot r^k = 0 \quad \text{and} \quad e_i^T \cdot s^k = 0 \quad \text{for } i = k+1, k+2, \ldots, l. \end{aligned} \tag{2.29}$$

The conditions (2.29) require that components $k+1$ through $m-k$ of r^k and s^k be zero, which are the non-Z-elements of r^k and s^k in M^k. That is, r^k and s^k have the form :

$$r^k = (r_1^k, \ldots, r_k^k, 0, \ldots, 0, r_l^k, \ldots, r_m^k)^T \tag{2.30}$$

$$s^k = (s_1^k, \ldots, s_k^k, 0, \ldots, 0, s_l^k, \ldots, s_m^k)^T . \tag{2.31}$$

Thus, an EULQ matrix of index k and order m is an ULQ matrix whose only nonidentity columns are columns k and l ($l = m-k+1$). In general, it has the form depicted in Figure 2.3.

The set of all nonsingular EULQ matrices is closed under inversion, and the inverse of any nonsingular EULQ matrix of index k is another EULQ matrix of index k.

Proposition 2.7

Let $M^{(k)} = M^k M^{k-1} \cdots M^1$ where M^i is an EULQ matrix of index i , i=1,2,...,k. Then $M^{(k)}$ is a ULQ matrix whose j^{th} and $(m-j+1)^{st}$ columns are those of M^j , $j=1,2,\ldots,m/2$. The proof is similar to that of Proposition 2.2.

Definition 2.4

A partially reduced left-right quadrant (PRLRQ) matrix of index k and order m is a square matrix whose non-W-elements are zero in columns $k+1$ through $m-k$. Note that $B^{m/2}$ has no special zero structure and B^1 is an LRQ matrix. In general, a PRLRQ matrix is of the form shown in Figure 2.4.

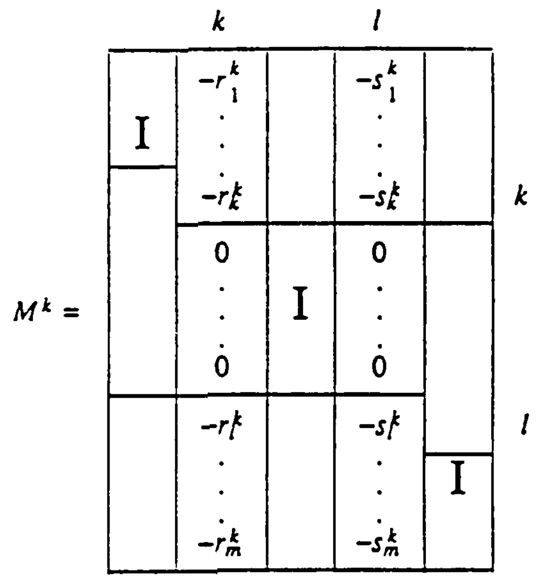


Figure 2.3. Illustration of the EULQ matrix of order m and index k.

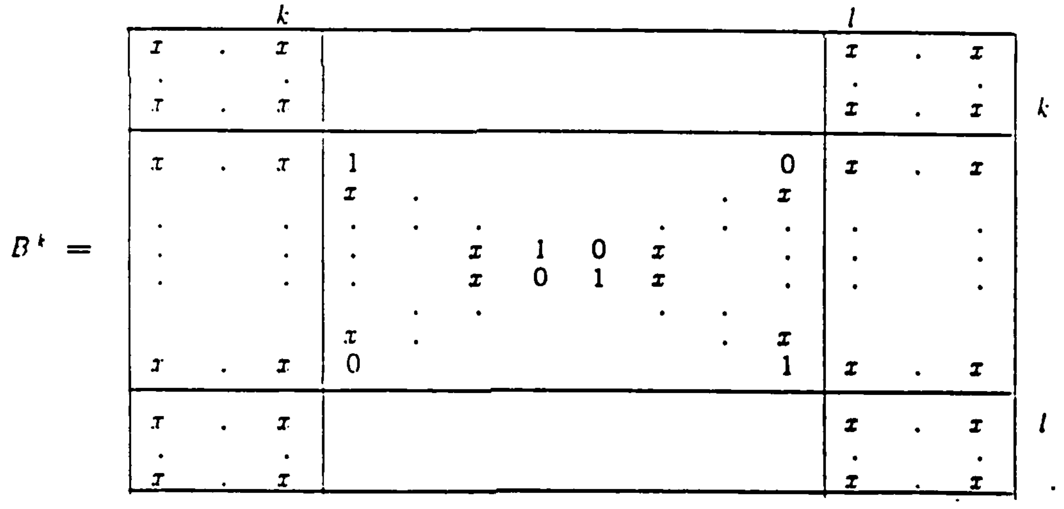


Figure 2.4. Illustration of a PRLRQ matrix of order m and index k.

Proposition 2.8

Let B^k be a PRLRQ matrix of index k. If B^k is nonsingular then there exist j_1 and j_2 such that $1 \le j_1 \le k$ and $l \le j_2 \le m$ and

$$\delta = B^k_{k,j_1} \cdot B^k_{l,j_2} - B^k_{k,j_2} \cdot B^k_{l,j_1} \neq 0 \tag{2.32}$$

The proof is similar to that of Proposition 2.3.

Now let j_1 and j_2 satisfy Proposition 2.8 and define P^k to be a permuted identity matrix with column j_1 in the k^{th} position and j_2 in the l^{th}. Let B^k be a nonsingular PRLRQ matrix of index k. Obviously, $\hat{B}^k = B^k P^k$ is a nonsingular PRLRQ matrix of index k , and $S^k(\hat{B}^k)$ is nonsingular.

Using M^k of (2.28) and the P^k defined above, the elimination operation needed to reduce a PRLRQ matrix of index k a step further is given by the following Proposition.

Proposition 2.9

Let B^k be a nonsingular PRLRQ matrix of index k , let j_1 and j_2 satisfy Proposition 2.8, let P^k be the permutation matrix that permutes columns k and j_1 and columns $m-k+1$ and j_2. Let M^k be an EULQ matrix of index k whose r^k,s^k vectors are determined by solving the following $2k-2$ linear systems

$$\begin{bmatrix} r_i^k & s_i^k \end{bmatrix} . S(\hat{B}^k) = \begin{bmatrix} \hat{B}^k_{i,k} & \hat{B}^k_{i,l} \end{bmatrix} \qquad , i=1,\dots,k-1 \text{ and } l+1,\dots,m \tag{2.33}$$

along with the system

$$\begin{bmatrix} r_k^k & s_k^k \\ r_l^k & s_l^k \end{bmatrix} = \begin{bmatrix} S^k(\hat{B}^k) \end{bmatrix}^{-1}. \tag{2.34}$$

Then $B^{k-1} = M^k B^k P^k$ is a nonsingular PRLRQ matrix of index $k-1$.

Given the above definitions, we may state the backward QIF algorithm as follows:

Algorithm 2.2 : The Backward Quadrant Interlocking Factorization

Let $B \in R^{m,m}$. The following steps decompose B to its QIF with $B = Z\ W\ P$.

```
Initialize
        $B^{m/2} = B$,
          $K = m/2$.
Main Loop
  For $k = K, K-1, K-2, \ldots, 1$
      1. Column Permutation
          Find $j_1$ and $j_2$ satisfying Proposition 2.8.
          If none exists, then terminate with the conclusion that $B$ is singular.
          Otherwise, construct $P^k$ using $j_1$ and $j_2$.
      2. Compute the vectors $r^k$, $s^k$
          by solving the $(2k-1)$ 2x2 linear systems (2.33) and (2.34).
      3. Construct $M^k$
                  $M^k = I_m - r^k \cdot e_k^T - s^k \cdot e_l^T - e_k \cdot e_k^T - e_l \cdot e_l^T$.
      4. Construct $B^{k-1}$
                  $B^{k-1} = M^k \; B^k \; P^k$.
  Next $k$
```

Proposition 2.10

Let B be a nonsingular matrix of size m. Then *Algorithm 2.2* decomposes B to its backward QIF,

$$B = Z \, W \, P \tag{2.35}$$

where

(1) $Z \varepsilon \{M_m^z\}$, $Z = (M^1 M^2 \cdots M^K)^{-1}$,
(2) $W \varepsilon \{M_m^w\}$, $W = B^1$,and
(3) P is a permutation matrix , $P = (P^K \cdots P^1)^{-1}$.

The proof is similar to that of Proposition 2.5.

As with the Forward QIF Algorithm, row and/or column permutations can be adopted to ensure numerical stability and/or sparse factors.

2.4 Some Characteristics of Quadrant Matrices

In this section we reveal a relationship between the quadrant and triangular matrices, which has not previously appeared in the open literature (e.g. Evans and Hatzopoulos [1979-1], Evans and Hadjidimos [1980-1], Evans [1982-1], Feilmeier [1982-1], Hellier [1982-1], and Shanehchi and Evans [1982-1]). A permutation algorithm that restructures any quadrant matrix as a block triangular one is presented.

Consider the following matrices

$$\hat{Z} = \begin{bmatrix} x & x & & & & & \\ x & x & & & & & \\ x & x & x & x & & & \\ x & x & x & x & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \\ x & x & x & x & & x & x \\ x & x & x & x & & x & x \end{bmatrix} , \quad \hat{W} = \begin{bmatrix} 1 & 0 & x & x & . & x & x \\ & 1 & x & x & . & x & x \\ & & 1 & 0 & . & x & x \\ & & & 1 & . & x & x \\ & & & & . & . & . \\ & & & & & . & . \\ & & & & & . & . \\ & & & & & 1 & 0 \\ & & & & & & 1 \end{bmatrix} . \qquad (2.36)$$

Where x stands for a potential nonzero element. Note that $\hat{Z}$ is a lower Hessenberg matrix with a special zero distribution on the superdiagonal. Also, $\hat{W}$ is a unit upper triangular matrix with special zero distribution on the superdiagonal.

Now we present an algorithm that relates W of (2.1) and Z of (2.3) to $\hat{W}$ and $\hat{Z}$ of (2.36).

Algorithm 2.3 : The Permutation Algorithm

Let R, S, and T be square matrices of order m, where R is the input matrix to the algorithm and T is the output matrix. The following algorithm permutes the columns and rows of R such that:

(a) if R is a LRQ matrix then T is a $\hat{W}$ of (2.36), and

(b) if R is a ULQ matrix then T is a $\hat{Z}$ of (2.36).

1. Column Permutation

For $j = 1, 2, \ldots, m/2$

```
        $S_{.,m-2j+1} \leftarrow R_{.,j}$
        $S_{.,m-2j+2} \leftarrow R_{.,m-j+1}$
    Next $j$
2. Row Permutation
    For $i=1,2,\ldots,m/2$
        $T_{m-2i+1,.} \leftarrow S_{i,.}$
        $T_{m-2i+2,.} \leftarrow S_{m-i+1,.}$
    Next $i$
```

An example of the permutation algorithm is given below for $m=6$.

Example 2.3 ($m=6$)

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ w_{2,1} & 1 & 0 & 0 & 0 & w_{2,6} \\ w_{3,1} & w_{3,2} & 1 & 0 & w_{3,5} & w_{3,6} \\ w_{4,1} & w_{4,2} & 0 & 1 & w_{4,5} & w_{4,6} \\ w_{5,1} & 0 & 0 & 0 & 1 & w_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \hat{W} = \begin{bmatrix} 1 & 0 & w_{3,2} & w_{3,5} & w_{3,1} & w_{3,6} \\ 0 & 1 & w_{4,2} & w_{4,5} & w_{4,1} & w_{4,6} \\ 0 & 0 & 1 & 0 & w_{2,1} & w_{2,6} \\ 0 & 0 & 0 & 1 & w_{5,1} & w_{5,6} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$Z = \begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\ 0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\ 0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\ 0 & 0 & z_{4,3} & z_{4,4} & 0 & 0 \\ 0 & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\ z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6} \end{bmatrix}, \hat{Z} = \begin{bmatrix} z_{3,3} & z_{3,4} & 0 & 0 & 0 & 0 \\ z_{4,3} & z_{4,4} & 0 & 0 & 0 & 0 \\ z_{2,3} & z_{2,4} & z_{2,2} & z_{2,5} & 0 & 0 \\ z_{5,3} & z_{5,4} & z_{5,2} & z_{5,5} & 0 & 0 \\ z_{1,3} & z_{1,4} & z_{1,2} & z_{1,5} & z_{1,1} & z_{1,6} \\ z_{6,3} & z_{6,4} & z_{6,2} & z_{6,5} & z_{6,1} & z_{6,6} \end{bmatrix}.$$

This clearly shows that the quadrant matrices are permuted block triangular matrices with blocks of size 2. That is, the Forward (Backward) Quadrant Interlocking factorization is equivalent to a block Doolittle (Crout) decomposition with blocks of size 2.

On sequential computers, a QIF is not expected to be faster than any triangular decomposition. Since computing the entries of the factors by solving 2x2 systems requires more operations, as shown in Proposition 2.6. Also, finding a nonsingular 2x2 submatrix is more expensive than finding a nonzero element. However, on parallel computers. the QIF is expected to be competitive, since the number of entries that can be produced concurrently in every stage is doubled, and the number of stages is halved as compared to a triangular factorization algorithm. Therefore, we may view the column permutation step in Algorithms 2.1 and 2.2 searching for a nonsingular 2x2 submatrix as a computation decoupling price we pay for the concurrency gained in steps 2-4.

Determining the relationship between quadrant and triangular matrices is a key observation that we will use in the following section to design appropriate updating scheme for the quadrant interlocking factors of the basis matrix in the simplex method.

III. UPDATING THE QIF OF THE BASIS

At the beginning of a simplex iteration, suppose the basis has the form

$$B = Z\,W\,R, \tag{3.1}$$

where we assume forms (2.36) for Z and W, and R is a permutation matrix. When the entering column $A_{.j}$ replaces the leaving column $B_{.p}$ at the end of the simplex iteration, we have a new basis matrix $\bar{B}$ which is related to the previous basis matrix B by the formula

$$\bar{B} = B\,E \tag{3.2}$$

where E is an eta matrix whose p^{th} column is $(B^{-1} A_{.j})$, and all other columns are the identity columns. From (3.1) and (3.2) $\bar{B}$ can be written as

$$\bar{B} = Z\,W\,R\,E. \tag{3.3}$$

An *updating scheme* is a sequence of operations applied to the right side of (3.3) to return it to the form given by (3.1), i.e.

$$\bar{B} = \bar{Z}\,\bar{W}\,\bar{R}, \tag{3.4}$$

where $\bar{W}$, $\bar{Z}$ are the new Q.I. factors and $\bar{R}$ is a permutation matrix. We present an algorithm designed to derive (3.4) given (3.3). It is similar to the Forrest-Tomlin [1972-1] update for the triangular factors of the basis. Since the spike is in W, our strategy is to reduce the spiked W, i.e., WE, to an LRQ matrix using elementary ULQ matrices. The following algorithm exploits the triangular form of W and the existence of 2x2 identity blocks on the diagonal of W.

In this presentation we use the term *brother columns (rows)* to indicate columns (rows) that have the same potential nonzero structure, excluding the diagonal entries in case of LRQ matrices. Thus, for LRQ matrices in the form of (2.36) columns (rows)

$i, i+1$ are brother columns (rows) for $i=1,3, \cdots, m-1$.

The first step of this scheme is a column permutation followed by a row permutation. In Figure 3.1 an example is presented to illustrate this step, in which R of (3.3) permutes columns 2 and 4 of W and x stands for potentially nonzero elements. Thus, W and WR are as illustrated in Figure 3.1 (a) and (b). From (3.3) we obtain

$$\begin{aligned} Z^{-1} \bar{B} &= W\, R\, E \\ &= \hat{S}, \end{aligned}$$

where $\hat{S}$ is illustrated in Figure 3.1 (c) and y stands for the elements of the column vector $(Z^{-1} A_{.,j})$. Note that if $(Z^{-1} A_{.,j})$ has the same zero structure as $W_{.,q}$, then the new factors are immediately available. That is, $\overline{W}$ is $\hat{S}$ and $\bar{Z}$ is Z. If this is not the case, we place $\hat{S}$ in a spiked-W form S as shown in Figure 3.1 (d), by applying the column permutation R^{-1} to $\hat{S}$ to undo the effect of R. That is,

$$\begin{aligned} Z^{-1} \bar{B}\, R^{-1} &= W\, R\, E\, R^{-1} \\ &= \hat{S}\, R^{-1} \\ &= S. \end{aligned} \tag{3.5}$$

Suppose $q < m-1$. We apply the column permutation $\hat{R}$ to S, placing the spike and the brother of the leaving column in the positions m and $m-1$, respectively, and moving all intervening columns forward to produce the matrix H^q, as illustrated in Figure 3.1(e). We then apply the row permutation $\hat{R}^{-1}$ to H^q placing the q^{th} row and its brother row in positions m and $m-1$, respectively, moving all intervening rows two places up to produce the matrix $H^{\bar{q}}$ as shown in Figure 3.1 (f), where

$$\bar{q} = \begin{cases} q, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

Note that $\bar{q}$ is odd. Of course, if $q \geq m-1$, then $\hat{R} = I$. Now (3.5) becomes

$$\begin{aligned} \hat{R}^{-1} Z^{-1} \bar{B}\, R^{-1} \hat{R} &= \hat{R}^{-1} W\, R\, E\, R^{-1} \hat{R} \\ &= \hat{R}^{-1} \hat{S}\, R^{-1} \hat{R} \\ &= \hat{R}^{-1} S\, \hat{R} \end{aligned}$$

1oxxxxxx	1xxoxxxx	1yxoxxxx	1oxyxxxx	1oxxxxxy	1oxxxxxy
o1xxxxxx	oxx1xxxx	oyx1xxxx	o1xyxxxx	o1xxxxxy	o1xxxxxy
oo1oxxxx	oo1oxxxx	oy1oxxxx	oo1yxxxx	ooxxxx1y	oo1oxxoy
ooo1xxxx	o1ooxxxx	oyooxxxx	oooyxxxx	ooxxxxoy	ooo1xxoy
oooo1oxx	oooo1oxx	oyoo1oxx	oooy1oxx	oo1oxxoy	oooo1ooy
ooooo1xx	ooooo1xx	oyooo1xx	oooyo1xx	ooo1xxoy	ooooo1oy
oooooo1o	oooooo1o	oyoooo1o	oooyoo1o	oooo1ooy	ooxxxx1y
ooooooo1	ooooooo1	oyooooo1	oooyooo1	ooooo1oy	ooxxxxoy
W	WR	$\hat{S}$	S	H^q	$H^{\bar{q}}$
(a)	(b)	(c)	(d)	(e)	(f)

Figure 3.1. Illustration of the double column and row permutation ($m=8, p=2, q=4, \bar{q}=3$).

```
 1                   l                     m

 l1 o x x        x x x x x x x x          x x x y    1
 lo 1 x x   ...  x x x x x x x x    ...   x x x y
    l1 o         x x x x x x x x          x x x y
    lo 1         x x x x x x x x          x x x y
             .          .                    .       .
              .         .                    .       .
              .         .                    .       .
                 l1 o x x x x x x         x x x y
                 lo 1 x x x x x x         x x x y
                     l1 o x x x x         x x o y    l
                     lo 1 x x x x   ...   x x o y
                         l1 o x x         x x o y
                         lo 1 x x         x x o y
                             l1 o         x x o y
                             lo 1         x x o y
                                    .        .       .
                                     .       .       .
                                     .       .       .
                                          l1 o o y
                                          lo 1 o y
                 lx xlx xlx x       ...   x xl1 y
                 lx xlx xlx x       ...   x xlo y    m .
```

Figure 3.2. Illustration of the general form of the matrix H^l.

$$= \hat{R}^{-1} H^{q}$$
$$= H^{\bar{q}}. \tag{3.6}$$

Consider the matrix H^l whose general form is depicted in Figure 3.2. Note that the matrix resulting from the above permutation is H^l when $l = \bar{q}$. Note also that all non-W-elements in H^l are in the last two rows in columns l through m. Our objective now is to reduce $H^{\bar{q}}$ to a LRQ matrix by eliminating these non-W-elements. We consider eliminating them four at a time using the 2x2 identities on the diagonal of H^l. The necessary matrices that should reduce H^l to H^{l+2}, for $l=\bar{q},\bar{q}+2, \cdots ,m-3$, are the following EULQ transformations.

$$Z^l = \begin{array}{|c|cc|c|}
\multicolumn{1}{c}{} & l-1 & \multicolumn{1}{c}{l} & \multicolumn{1}{c}{} \\
\hline
 & 0 & 0 & \\
I & \vdots & \vdots & \\
 & 0 & 0 & \\
\cline{1-3}
 & 1 & 0 & \\
 & 0 & 1 & \\
\cline{2-4}
 & 0 & 0 & \\
 & \vdots & \vdots & I \\
 & 0 & 0 & \\
\cline{2-3}
 & -H^l_{m-1,l-1} & -H^l_{m-1,l} & \\
 & -H^l_{m,l-1} & -H^l_{m,l} & \\
\hline
\end{array}$$

By repetitive application of Z^l to H^l, for $l = \bar{q},\bar{q}+2, \cdots ,m-3$, we get H^{m-1} which, in general, has a non-W-element in its $m-1,m$ entry and a nonconforming element in the m,m^{th} entry. Therefore, the following rank-one elementary transformation is sufficient to reduce H^{m-1} to the LRQ matrix $\overline{W}$,

$$Z^{m-1} = \begin{array}{|c|c|c} \multicolumn{1}{c}{} & \multicolumn{1}{c}{m} & \\ \hline I & & \\ \hline & -H^{m-1}_{m-1,m} / H^{m-1}_{m,m} & m-1 \\ & 1 / H^{m-1}_{m,m} & m \\ \hline \end{array}$$

Theoretically, $H^{m-1}_{m,m}$ is a nonzero element, since otherwise $\bar{B}$ is singular. Now, combining all transformations applied to $H^{\bar{q}}$, we obtain,

$$Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}} = \overline{W},$$

and (3.6) becomes,

$$\{Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} \hat{R}^{-1} Z^{-1}\} \bar{B} \{R^{-1} \hat{R}\} = Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}} \qquad (3.7)$$

$$\{\bar{Z}^{-1}\} \quad \bar{B} \quad \{\bar{R}^{-1}\} = \overline{W},$$

which is equivalent to the required updated form (3.4), with

$$\bar{Z} = Z \hat{R} Z^{\bar{q}^{-1}} \cdots Z^{m-3^{-1}} Z^{m-1^{-1}}, \qquad (3.8)$$

$$\bar{R} = R^{-1} \hat{R}, \text{ and}$$

$$\overline{W} = Z^{m-1} Z^{m-3} \cdots Z^{\bar{q}} H^{\bar{q}}.$$

Note that $\bar{Z}$ in (3.8) is not a ULQ matrix, even though all its factors, except the permutation matrix $\hat{R}$, are ULQ matrices. In practice, $\bar{Z}^{-1}$ is stored factorized as in the first braced term in the left hand side of (3.7).

Using the above, the updating algorithm may be stated as follows:

Algorithm 3.1 : The B.Q.I.F. Updating Algorithm

0. Begin with the m x m matrix $B = Z\ W\ R$, and suppose column p of B is replaced by $A_{.,j}$.
1. Define q such that $R_{.p} = e_q$.

2. Set

$$S_{.,i} \leftarrow \begin{cases} Z^{-1} A_{.,j},\ i=q; \\ W_{.,i},\ \textit{otherwise}. \end{cases}$$

3. Let

$$\hat{q} = \begin{cases} q+1, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

4. Set

$$\hat{R}_{.,i} \leftarrow \begin{cases} e_i,\ 1 \le i < q; \\ e_{i+2},\ q \le i < m-1; \\ e_{\hat{q}},\ i = m-1; \\ e_q,\ i = m. \end{cases}$$

5. $H \leftarrow \hat{R}^{-1} S \hat{R}$.

6. Let

$$\bar{q} = \begin{cases} q, \text{ if } q \textit{ is odd}; \\ q-1, \text{ if } q \textit{ is even}. \end{cases}$$

For $l=\bar{q}, \bar{q}-2, \cdots, m-3$.

7. Set

$$Z^l_{i,l} \leftarrow \begin{cases} 1,\ i=l; \\ -H_{m-1,l},\ i=m-1; \\ -H_{m,l},\ i=m; \\ 0,\ \textit{otherwise}. \end{cases}$$

$$Z^l_{i,l+1} \leftarrow \begin{cases} 1,\ i=l+1; \\ -H_{m-1,l+1},\ i=m-1; \\ -H_{m,l+1},\ i=m; \\ 0,\ \textit{otherwise}. \end{cases}$$

$$Z_{.,j} \leftarrow e_j,\ j \ne l \text{ and } j \ne l+1.$$

8. $H \leftarrow Z^l H$.

Next l.

9. Set

$$Z^{m-1}_{i,m} \leftarrow \begin{cases} -H_{m-1,m}/H_{m,m}, \; i=m-1; \\ 1/H_{m,m}, \; i=m; \\ 0, \; otherwise. \end{cases}$$

$$Z^{m-l}_{\cdot,j} \leftarrow e_j, \; j \neq m.$$

10. $H \leftarrow Z^1 H$.

11. Set

$$\bar{B} = \{Z \hat{R} (Z^{\bar{q}})^{-1} \cdots (Z^{m-1})^{-1}\} H \{\hat{R}^{-1} R\}$$

$$= \bar{Z} \bar{W} \bar{R}.$$

This updating scheme inherits the major characteristics of the Forrest-Tomlin update for the triangular factors of the basis. First, no new nonzeros are created in the right factor W, since only deletions of items are required. Therefore, sparsity of W is preserved and fill-in is minimized. Second, the lack of choice of the pivot elements makes this update less numerically stable than the Bartels-Golub-based updates. Thus, there is a gain in speed and storage at some sacrifice in numerical stability.

IV. PARALLEL IMPLEMENTATION

In this section we describe a parallel implementation of two basic tasks of any simplex based linear programming code, namely, basis reinversion and solution of the linear systems. A parallel version of the Backward Quadrant Interlocking Factorization Algorithm (BQIF) is presented in Section 4.1. Only the left factor is produced in its product form while the right factor is produced in its explicit form. This form conforms with the updating scheme of Section III. In this algorithm, parallelism is gained by reformulating the BQIF Algorithm in terms of high-level modules such as matrix-vector operations. These modules represent a high level of granularity in the algorithm in the sense that they are based on matrix-vector operations, $O(m^2)$ work, not just vector operations, $O(m)$ work. The module concept has already proven to be very successful in achieving both transportability and high performance of some linear algebra routines across a wide range of architectures, as reported by Dongarra and Sorensen [1984-2] and Dongarra and Hewitt [1986-1].

Given a basic feasible solution with basis B, each iteration of Dantzig's simplex algorithm involves solving the systems of equations $\pi B = c^B$ and $B\,y = A_{.,j}$. An efficient parallelization of the simplex algorithm requires efficient parallel algorithms for solution of these systems. Parallel algorithms for solving these linear systems using the quadrant factors are presented in Section 4.2. The parallel implementation discussed in this section is proposed for an MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of *multitasking*, that is, the processors are capable of applying all their power to a single job in a timely and coordinated manner.

4.1 The Module-Based BQIF Algorithm

Given an m x m matrix B, the algorithm either indicates singularity of B or produces

$$Z^{m-1} Z^{m-3} \cdots Z^{1} B\, R = W, \tag{4.1}$$

where R is a permutation matrix, Z^k is a rank-2 matrix of the form,

$$Z^k = \begin{array}{|c|c|c|c|l}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{k} & \multicolumn{1}{c}{k+1} & \multicolumn{1}{c}{} & \\
\hline
\mathrm{I} & & & & \\
\hline
 & x & x & & k \\
 & x & x & & k+1 \\
\hline
 & x & x & & \\
 & \vdots & \vdots & \mathrm{I} & \\
 & x & x & & \\
\hline
\end{array}, \tag{4.2}$$

This form conforms with the updating schemes of Section III. Its LU version has been used in several LP codes (Reid [1982-1]) . At every stage a new Z^i is produced and two rows of W are updated. The availability of the updated rows of W at every stage allows for parallel implementation when searching for a nonsingular 2x2 submatrix. Moreover, it facilitates finding the 2x2 submatrix of largest determinant rather than finding one with a nonzero determinant. This reduces the rounding error in the factorization process and hence improves the numerical accuracy of the results (Shanehchi and Evans [1982-1]).

The major part of the algorithm is formulated in terms of three basic modules:

Module 1 : *Search for a nonsingular 2x2 submatrix*

Input : $A \in R^{2,n}$

Purpose : Find column indices j_1 and j_2 such that

$$DET = A_{1j_1} \cdot A_{2j_2} - A_{2j_1} \cdot A_{1j_2} \neq 0.$$

Output : j_1, j_2, and DET or a singular indication.

Module 2 : Matrix - 2 vectors product

Input : $y^1 \varepsilon R^{n_1,2}$, $A^1 \varepsilon R^{n_1,n_2}$, $x^1 \varepsilon R^{n_2,2}$.

Purpose : Compute y^1 such that $y^1 \leftarrow y^1 + A^1 x^1$.

Output : y^1.

Module 3 : 2 vectors - matrix product

Input : $y^2 \varepsilon R^{2,l_2}$, $x^2 \varepsilon R^{2,l_1}$, $A^2 \varepsilon R^{l_1,l_2}$.

Purpose : Compute y^2 such that $y^2 \leftarrow y^2 + x^2 A^2$.

Output : y^2.

These modules represent a high level of granularity in the algorithm in the sense that they are based on matrix-vector operations, $O(m^2)$ work, not just vector operations, $O(m)$ work.

Algorithm 4.1 : Reinversion

Let $B \varepsilon R^{m,m}$. Then the following steps produce a singular indication if B is singular, or decompose B as in (4.1) if B is nonsingular. The column indices are stored in $IPVT(m)$. Define the 2x2 submatrix $S_{i,j}(A)$ to be

$$S_{i,j}(A) = \begin{bmatrix} A_{i,j} & A_{i,j+1} \\ A_{i+1,j} & A_{i+1,j+1} \end{bmatrix}. \qquad (4.3)$$

0. Initialize.

$$W_{1:2,1:m} \leftarrow B_{1:2,1:m}$$

For $i = 1,3, \cdots, m-3$

```
  1. Find a nonsingular 2x2 submatrix.
     Set $n \leftarrow m - i + 1$ and $A \leftarrow W_{i:i+1,i:m}$.
       Call Module 1 $(A, n)$.
     If $A$ is singular, then terminate with $B$ singular;
```

```
        otherwise, permute columns
            $W_{1:m,i}$ with $W_{1:m,j_1}$ and $W_{1:m,i+1}$ with $W_{1:m,j_2}$.
        Record permutation, $IPVT(i)=j_1$, $IPVT(i+1)=j_2$.

    2. Obtain a new $Z^i$.
        $Z^i \leftarrow I$, where $I$ is m x m, $S_{i,i}(Z^i) \leftarrow [S_{i,i}(W)]^{-1}$.
        $n_1 \leftarrow m-i-1$, $n_2 \leftarrow i-1$.
        $y^1 \leftarrow B_{i+2:m,i:i+1}$, $x^1 \leftarrow W_{1:i-1,i:i+1}$.
        For $l = 1,3, \cdots, i-2$
            $A^1_{.,l:l+1} \leftarrow Z^l_{i+2:m,l:l+1}$.
        Next $l$.
        Call Module 2 $(y^1, x^1, A^1, n_1, n_2)$
        $Z^i_{i+2:m,i:i-1} \leftarrow y^1$.

    3. Update rows $i+2$, $i+3$ of $W$.
        $l_1 \leftarrow i+1$, $l_2 \leftarrow m-i+1$.
        $A^2 \leftarrow W_{1:i+1,i+2:m}$, $y^2 \leftarrow B_{i+2:i+3,i+2:m}$.
        For $l=1,3, \cdots, i$
            $x^2_{.,l:l+1} \leftarrow Z^l_{i+2:i+3,l:l+1}$.
        Next $l$.
        Call Module 3 $(y^2, x^2, A^2, l_1, l_2)$
        $W_{i+2:i+3,i+2:m} \leftarrow y^2$.

Next $i$.

4. Update $W$.
For $i = 1,3, \cdots, m-3$
    $S_{i,i}(W) \leftarrow I$, where $I$ is 2 x 2.
    $W_{i:i+1,i+2:m} \leftarrow S_{i,i}(Z^i)\, W_{i:i+1,i+2:m}$
```

Next i.

The general approach we propose for parallel implementation involves having the parent processor prepare the parameters for a module and make use of the kids (subtask processors) to work concurrently on that module. In Module 1, at most $n(n-1)/2$ column pairs should be checked. The parent sends to each kid the column indices to be checked for nonsingularity, and stops all kids whenever one succeeds. As mentioned before, it is possible to find the nonsingular 2x2 submatrix of largest determinant. To do this, the parent sends the column indices to the kids, each kid finds the column pair of largest determinant in his list, sends them to the parent, then the parent selects the best by comparing only $p-1$ values.

The concurrency in Modules 2 and 3 is obvious since they involve matrix-vector operations. In Module 2 (matrix - 2 vectors product) parallelism is obtained by performing $2n_1$ independent inner products, where n_1 is the row dimension of the matrix. Similarly, in Module 3 (2 vectors - matrix product) concurrency is gained by executing $2l_2$ independent inner products, where l_2 is number of columns of the matrix. Step 3 needs only $Z^i_{i+2,i+3,\ldots,n+1}$ from Step 2. These are the first two rows of y^1. Thus, as soon as these elements become available Step 3 may proceed. This can easily be synchronized. Finally, in Step 4 the loop divides over i with completely independent tasks. However, the tasks require different amounts of computation. Two solutions are possible. Either we adopt dynamic task queue allocation, or we statically allocate $i=1, m-3$ to one processor, $i=3, m-5$ to the second, and so on.

4.2 Solving the Linear Systems

In this section we investigate the possible parallelism involved when we solve the systems of equations $\pi B = c^B$ and $B\,y = A_{.j}$. We assume that the basis matrix B is in the form (4.1), that is

$$Z^{m-1} Z^{m-3} \cdots Z^1 B\, R = W,$$

where Z^k has the form (4.2), and W is a block unit upper triangular matrix with blocks of size 2.that is it has the form (2.36). We compute the dual variables (π) using the following steps:

(1) Permutation : $\bar{\pi} = c^B R$.

(2) Solve a block triangular system : $\hat{\pi} W = \bar{\pi}$.

(3) BTRAN : $\pi = \hat{\pi} Z^{m-1} Z^{m-3} \cdots Z^1$.

We compute y, the basis representation of the incoming column $A_{.,j}$, as follows:

(1) FTRAN : $\hat{y} = Z^{m-1} Z^{m-3} \cdots Z^1 A_{.,j}$.

(2) Solve a block triangular system : $W\, \bar{y} = \hat{y}$.

(3) Permutation : $y = R\, \bar{y}$.

We present parallel implementations of the FTRAN operation, the solution of a block triangular system, and the BTRAN operation in Sections 4.2.1, 4.2.2, and 4.2.3, respectively.

4.2.1 The FTRAN in Parallel

The rules for applying a Z^k to an arbitrary vector v are as follows:

a) Extract $\alpha_k \leftarrow v_k$, and $\alpha_{k+1} \leftarrow v_{k+1}$.

b) Set $v_k \leftarrow 0$, and $v_{k+1} \leftarrow 0$.

c) Compute $\bar{v} = v + \alpha_k Z^k_{k:m,k} + \alpha_{k+1} Z^k_{k:m,k+1}$.

Note that if $v_k = v_{k+1} = 0$, then $\bar{v} = v$ and no element of v will change.

An example is now given for $m = 6$, $k = 3$. Suppose we have

$$Z^3_{.,3:4} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 1 & 2 \\ 1/3 & 1/4 \\ 1/6 & 1/2 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \text{ and } \quad u = \begin{bmatrix} 11 \\ 12 \\ 0 \\ 0 \\ 15 \\ 16 \end{bmatrix}.$$

Then the computation of Z^3v is given by

$$Z^3 v = \begin{bmatrix}1\\2\\0\\0\\5\\6\end{bmatrix} + 3\begin{bmatrix}0\\0\\2\\1\\1/3\\1/6\end{bmatrix} + 4\begin{bmatrix}0\\0\\1\\2\\1/4\\1/2\end{bmatrix} = \begin{bmatrix}1\\2\\10\\11\\7\\8.5\end{bmatrix},$$

and the computation of Z^3u is given by

$$Z^3 u = \begin{bmatrix}11\\12\\0\\0\\15\\16\end{bmatrix} + 0\begin{bmatrix}0\\0\\2\\1\\1/3\\1/6\end{bmatrix} + 0\begin{bmatrix}0\\0\\1\\2\\1/4\\1/2\end{bmatrix} = \begin{bmatrix}11\\12\\0\\0\\15\\16\end{bmatrix}.$$

These rules are implemented in the following module:

Module : FTRAN Operation (A, v, n)

Purpose : Apply Z^k to an arbitrary vector v.

Input : n, $A \in R^{n,2}$, $v \in R^{n,1}$.

Output : v, where $v = Z^k v$.

```
Steps : 1. Extract $\alpha_1 \leftarrow v_1$, and $\alpha_2 \leftarrow v_2$.
        2. Set $v_1 \leftarrow 0$, and $v_2 \leftarrow 0$.
        3. Compute $A_{.,1} \leftarrow \alpha_1 A_{.,1}$.
        4. Compute $A_{.,2} \leftarrow \alpha_2 A_{.,2}$.
        5. Compute $v \leftarrow v + A_{.,1} + A_{.,2}$.
```

Obviously, steps 3 and 4 are independent and can be executed in parallel. In step 5, the work is partitioned over the rows of v, assigning each kid a block of rows to evaluate.

4.2.2 Solving the Block Triangular System

The solution of an m x m triangular system of equations on a sequential computer can be obtained by either a forward or backward substitution process which requires $O(m^2)$ steps, each defined as one multiplication followed by one addition. In order to solve the system on a parallel computer, methods which require $O(m^3)$ processors and, hence, reduce the computation time to $O(\log^2 m)$ have been developed (e.g. Chen and Kuck [1975-1] and Sameh and Brent [1977-1]). Evans and Dunbar [1983-1] introduced methods that run in $O(m)$ time using $O(m)$ processors. For practical purposes the processor and storage requirement of these methods is unreasonably large.

In this subsection we consider solving the linear system

$$xW = b, \tag{4.4}$$

where $x, b \in R^m$ and W is an upper triangular m x m matrix with 2x2 identity diagonal blocks. This system may be solved by a forward substitution (FS) process described in algorithmic form as follows.

```
For $i = 1,2, \cdots ,m$
    $x_i = b_i - \sum_{j=1}^{i-1} W_{i,j} x_j .$
Next $i$.
```

It is obvious that a uniprocessor will solve (4.4) sequentially in $m(m-2)/2$ steps by the FS process. Let T_p denote the time required to solve (4.4) using p processors, where one step requires one unit of time. Then

$$T_1 = m\ (m-2)/2.$$

With a parallel computer that has p processors, a minimum time requirement for the solution of (4.4) is

$$\min(T_p) = T_1/p = m\ (m-2)/(2p). \tag{4.5}$$

The minimum completion time of any algorithm based on FS is equal to the number of terms in the expression that evaluates x_m, that is

$$T_{\min} = m - 2.$$

From (4.5) it is clear that a minimum of $m/2$ processors is necessary to solve (4.4) in the minimum time of $m-2$ operations. Again this processor requirement is unreasonably large for our application.

The machine we consider has a limited number of identical processors ($p \le 30$). Therefore, we consider the question: if we are given a fixed number of processors, how should the parallel operations be scheduled on the processors to minimize the solution time of (4.4)? We propose to answer this question using a directed graph model that represents the FS process as follows. The nodes of the graph represent tasks of equal execution time and the edges represent the precedence relationships between the tasks. Then we apply a simple scheduling algorithm due to Hu [1961-1], called the *level algorithm*, to schedule the tasks on the processors such that the total execution time is minimized. This algorithm is known to be optimum for a tree graph, and it gives extremely good results for general graphs as reported by Ramamoorthy et al [1972-1], Huang and Wing [1979-1], and Wing and Huang [1980-1].

We first organize the FS process in terms of operations of equal time and define the corresponding directed graph. Let $x^i = [x_i, x_{i+1}]$. Partition x, b, and W into blocks of size 2. Using $S_{i,j}$ as defined in (4.3), the above FS process can then be written as

```
For $i = 1,3,\cdots,m-1$
  $x^i = b^i - \sum_{j=1,3,\cdots,i-2} x^j S_{i,j}(W)$.
Next $i$.
```

Let the following operation, where x^i is used to update x^j, define a task

$$x^j \leftarrow x^j - x^i S_{i,j}(W). \qquad (4.5)$$

For Hu's algorithm we assume that the execution time of an operation (4.5) is one unit (4 multiplications and 4 additions). We can see that the FS process consists of a set of operations (4.5), on which a set of precedence relations exists. That is, to complete the evaluation of x^i we require x^{i-2}, for $i = 3,5, \cdots ,m-1$. The process can therefore be represented by a directed graph $G(V,E)$ where the vertex set V is defined as

$$V \equiv \{v_{i,j} \mid v_{i,j} \textit{ represents an operation } (4.6)\},$$

and the edge set E is defined as

$$E \equiv \{(v_{i,j}\ , v_{k,l}) \mid \textit{operation } v_{k,l} \textit{ requires the direct result of operation } v_{i,j}\}.$$

We shall call $G\ (V,E)$ the *forward substitution task graph,* and refer to it by FSTG. In Figure 4.1 the FSTG for $m=10$ is presented. For every $v_{i,j}$ in the FSTG, the pair i,j is indicated. A node is an *initial node* if it does not have a predecessor and is a *terminal node* if it has no successor. It is clear that the FSTG has only one terminal node, at which $i = m-3$ and $j = m-1$. Accordingly, the minimum completion time, denoted by D, of the FSTG is equal to the number of nodes on the longest path from an initial node to the terminal node. Thus, $D = (m/2) - 1$, which is the number of times operation (4.6) is executed for x^{m-1}.

We next determine the levels of the vertices of the FSTG. Define the *level number* $(l_{i,j})$ of a node $v_{i,j}$ as follows: 1) the level of the terminal node is D, 2) the level of a node that has one or more successors is equal to the minimum of the levels of its successors minus one. Applying this definition to the FSTG, we can conclude that

$$l_{i,j} = (\ i+1\ )\ /\ 2. \tag{4.6}$$

The level number is simply the latest time by which node $v_{i,j}$ must be processed in order to complete the task graph in the minimum time D. The level numbers of the nodes of Figure 4.1 are given as shown.

Once the level numbers of the operations are determined, we apply Hu's scheduling

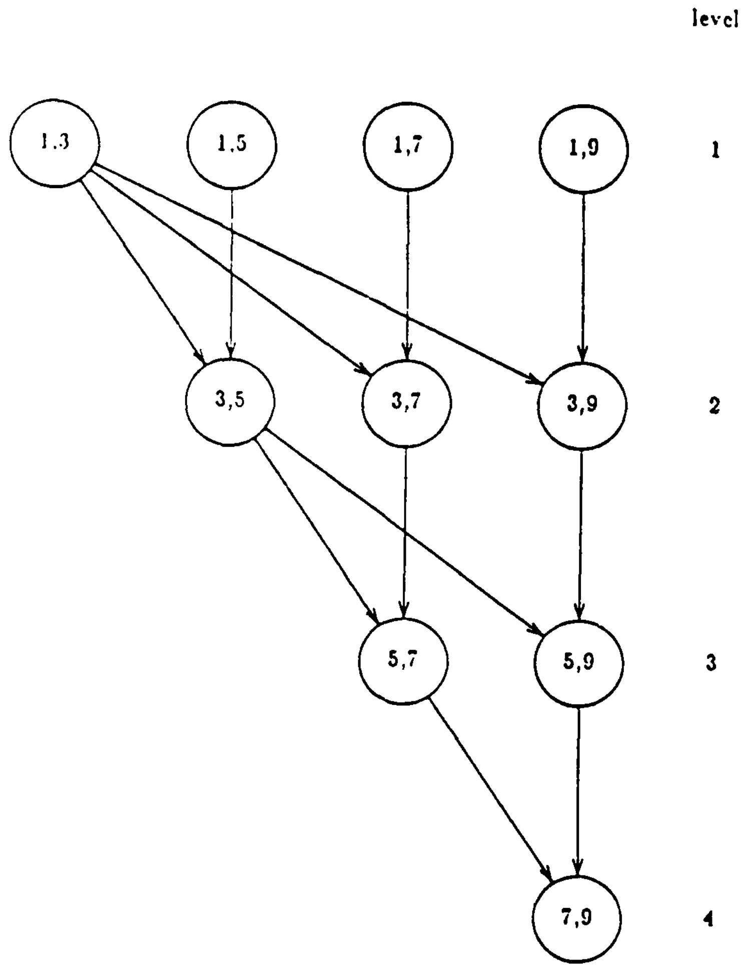


Figure 4.1. The FSTG for $m = 10$.

algorithm to assign operations to processors. Define a ***ready task*** to be one whose immediate predecessors have all been processed. The scheduling algorithm is as follows.

Algorithm 4.2: Hu's Scheduling Algorithm

1. Among all the ready tasks, schedule the one with smallest level number.

2. If there is a tie, schedule the one with the largest number of immediate successors.

Applying this Algorithm to the FS process represented by FSTG, the computations are organized as follows.

Algorithm 4.3 : Forward Substitution

```
Set $x^1 \leftarrow b^1$.
For $k = 3,5,\cdots,m-1$
  $x^k \leftarrow b^k - x^1 S_{1,k}(W)$.
Next $k$.
For $i = 3,\cdots,m-3$
  For $j = i+2,i+4,\cdots,m-1$
      $x^j \leftarrow x^j - x^i S_{i,j}(W)$.
  Next $j$.
Next $i$.
```

All operations in loop k are independent and have the same level number. Their level number ($l_{1,k} = 1$) is the smallest among all other operations in the Algorithm, and hence they are executed first. Similarly, all operations in loop j are independent and have the same level number as given by (4.6). The ordering of index i predicates the execution of the operations by increasing level number. This satisfies the first criterion in Hu's Algorithm. The second criterion imposes the ordering of the index j. That is, the number of immediate successors of $v_{i,j}$ is always greater than or equal to that of $v_{i,j+2}$ for

$j = i+2, i+4, \cdots, m-1$.

A parallel implementation of Algorithm 4.3 involves having the parent processor partition the work in loop k among the kids. Then for every i, the computational tasks of loop j are again divided among the kids.

Lower bounds on the completion time of a task graph given a fixed number of processors were derived by Ramamoorthy et al. [1972-1]. Let n_k be the number of nodes in level k. Let $t^*(p)$ be the minimum completion time to process a task graph with p processors. Then

$$t^*(p) \geq \max_i \left\lceil \frac{\sum_{k=1}^{i} n_k}{p} + D - i \right\rceil, \tag{4.7}$$

where D is the minimum completion time of the task graph and $\lceil x \rceil$ denotes the smallest integer $\geq x$. The first term in the expression denotes the minimum number of time units required to complete all the operations of the first i levels using p processors. The term $D-i$ is equal to the number of remaining levels yet to be processed. This bound may be useful in demonstrating optimality of the scheduling using Hu's Algorithm.

4.2.3 Parallel Implementation of the BTRAN Operation

In this section, we consider the parallel implementation of the following operation

$$\pi = \hat{\pi} Z^{m-1} Z^{m-3} \cdots Z^1,$$

where $\hat{\pi}$ is an arbitrary vector of m elements and each Z^k is an m x m rank-2 matrix that has the form (4.3).

The rule for computing $\bar{u} = u\ Z^k$ is as follows:

a) Set $\bar{u}_i \leftarrow u_i$ for $i \neq k$ and $i \neq k+1$.

b) Set $\bar{u}_k \leftarrow u_{k:m}\ Z^k_{k:m,k}$.

c) Set $\bar{u}_{k+1} \leftarrow u_{k:m}\ Z^k_{k:m,k+1}$.

For example, let $m = 6$, $k = 3$ and suppose we have

$$Z^3_{:,3:4} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 1 \\ 1 & 2 \\ 3 & 4 \\ 6 & 2 \end{bmatrix}, \text{and} \quad u = [\,1\ 1\ 1\ 1\ 1\ 1\,].$$

Then $\bar{u} = u\ Z^k = [\ 1\ \ 1\ \ 12\ \ 9\ \ 1\ \ 1\ ]$.

Note that $\bar{u}$ differs from u in only the k^{th} and the $k+1^{st}$ elements. Note also that the elements u_i, $i=1, \cdots, k-1$, are not required in computing $\bar{u}$. Using these observations, the BTRAN process may be represented by the following.

```
For $k = m-1, \cdots, 1$
  $u_k \leftarrow u_{k:m}\ Z^k_{k:m,k}$.
  $u_{k+1} \leftarrow u_{k:m}\ Z^k_{k:m,k-1}$.
Next $k$.
```

We now apply the methodology stated at the end of the previous subsection. Let the following operations define a task

$$u^k \leftarrow u^k\ S_{k,k}(Z^k). \tag{4.8}$$

$$u^j \leftarrow u^j + u^i\ S_{i,j}(Z^j). \tag{4.9}$$

We assume that the execution time of both operations is one unit. The task graph $G(V,E)$ of the BTRAN process is defined by the vertex set V, where

$$V \equiv \left\{ v_{i,j} \mid v_{i,j} \textit{ represents} \left\{ \begin{array}{l} \textit{an operation } (4.8), \text{ if } i=j; \\ \textit{an operation } (4.9), \textit{ otherwise} \end{array} \right\} \right\},$$

and the edge set E, where

$$E \equiv \{(v_{i,j}, v_{k,l}) \mid \textit{operation } v_{k,l} \textit{ requires the direct result of operation } v_{i,j}\}.$$

$G(V,E)$ has only one terminal node at which $i = 3$ and $j = 1$. Following the same arguments used earlier with FSTG, we conclude that

$$D = m / 2,$$

and

$$l_{i,j} = \begin{cases} 1, \text{if } i = j; \\ (m - i + 3) / 2, \textit{otherwise.} \end{cases}$$

Applying Hu's Algorithm to the BTRAN task graph yields the following ordering of computations.

Algorithm 4.4 : BTRAN Operation

```
For $k = m-1, m-3, \cdots, 1$
  $u^k \leftarrow u^k S_{k,k}(Z^k)$.
Next $k$.
For $i = m-1, m-3, \cdots, 3$
  For $j = i-2, i-4, \cdots, 1$
      $u^j \leftarrow u^j + u^i S_{i,j}(Z^j)$.
  Next $j$.
Next $i$.
```

The ordering of the index i is imposed by the first criterion of Hu's Algorithm. The ordering of the indices k and j is the result of applying the second criterion. Parallelism is gained by having the kid processors work first on loop k in parallel, and then for every i, having the kid processors work on loop j in parallel.

V. SUMMARY

Evans and Hatzopoulos [1979-1] developed a new matrix factorization, known as *the Quadrant Interlocking Factorization (QIF)*, for solving linear systems on parallel computers. In this paper we have presented the algorithms required to use this new factorization in Dantzig's simplex algorithm for linear programming. This work may be viewed as a parallelization of the simplex method using a quadrant interlocking factorization for the basis inverse.

In Section II, the factorization algorithms are developed, and the relationship of quadrant and triangular matrices is presented. In Section III, a new algorithm is presented for updating the factorization during a basis exchange step. In Section IV, we present a parallel implementation of the factorization algorithm, and develop the algorithms required to solve the linear systems of the simplex method on a parallel computer using the QIF of the basis. For each algorithm the concurrency among the steps is revealed, the computations are organized and a parallel implementation is proposed. The algorithms are designed for an MIMD parallel computer that incorporates p identical processors sharing a common memory and capable of applying all their power to a single application in a timely and coordinated manner.

REFERENCES

Bartels. R. H., 1971-1, "A Stabilization of the Simplex Method," Numer. Math., 16, pp. 414-434.

Chen. S. C., and D. Kuck, 1975-1, "Time and Parallel Processor Bounds for Linear Recurrence Systems," IEEE Trans. Comput., C-24, pp. 101-117.

Chen, S. S., J. J. Dongarra and C. C. Hsiung, 1984-1, "Multiprocessing Linear Algebra Algorithms on the CRAY X-MP-2: Experiences with Small Granularity," J. Parallel and Distributed Computing, 1, pp. 22-31.

Dongarra. J. J., A. H. Sameh and D. C. Sorensen, 1984-1, "Implementation of Some Concurrent Algorithms for Matrix Factorization," Argonne Nat. Lab., Argonne, IL, Rep. ANL/MCS-TM-25.

_____, and D. C. Sorensen, 1984-2, "A Parallel Linear Algebra Library for the Denelcor HEP," Argonne Nat. Lab., Argonne, IL, Rep. ANL/MCS-TM-33.

_____, and T. Hewitt. 1986-1, "Implementing Dense Linear Algebra Algorithms Using Multitasking on the CRAY X-MP-4," SIAM J. Sci. Stat. Comput., 7, pp. 347-350.

Evans. D. J., and M. Hatzopoulos, 1979-1, "A Parallel Linear System Solver," Intern. J. Computer Math., 7, pp. 227-238.

_____, and A. Hadjidimos, 1980-1, "A Modification of the Quadrant Interlocking Factorisation Parallel Method," Intern. J. Computer Math., 8, pp. 149-166.

_____, 1982-1, "Parallel Numerical Algorithms for Linear Systems," in Parallel Processing Systems, (D. J. Evans, ed.), Cambridge Univ. Press, Cambridge , pp. 357-384.

_____, and R. C. Dunbar, 1983-1, "The Parallel Solution of Triangular Systems of Equations." IEEE Trans. Comput., C-23, pp. 201-204.

Feilmeier. M., 1982-1, "Parallel Numerical Algorithms," in Parallel Processing Systems, (D. J. Evans, ed.), Cambridge University Press, Cambridge , pp. 285-338.

Forrest. J. J. H., and J. A. Tomlin, 1972-1, "Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method," Mathematical Programming, 2, pp. 263-278.

Hellier. R. L., 1982-1, "DAP Implementation of the WZ Algorithm," Comp. Phys. Comm., 26, pp. 321-323.

Huang, J. W., and O. Wing, 1979-1, "Optimal Parallel Triangulation of a Sparse Matrix." IEEE Trans. Circuits Syst., CAS-26, pp. 726-732.

Hu, T. C., 1961-1, "Parallel Sequencing and Assembly Line Problems," Operations Research, 9, pp.841-848.

Markowitz, H. M., 1957-1, "The Elimination Form of the Inverse and its Application to Linear Programming," Management Science, 3, pp. 255-269.

Ramamoorthy, C. V., K. M. Chandy and M. J. Gonzalez, 1972-1, "Optimal Scheduling Strategies in a Multiprocessor System," IEEE Trans. Comput., C-21, pp. 137-146.

Reid, J. K., 1982-1, "A Sparsity Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases," Math. Programming, 24, pp. 55-69.

Sameh, A. H., and R. P. Brent, 1977-1, "Solving Triangular Systems on a Parallel Computer," SIAM J. Numer. Anal., 14, pp. 1101-1113.

Saunders, M. A., 1976-1, "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating," in Sparse Matrix Computations, (J. R. Bunch and D. J. Rose, eds.), Academic Press, New York, New York, pp. 213-226.

Shanehchi, J. and D. J. Evans, 1982-1, "Further Analysis of the Quadrant Interlocking Factorisation (Q.I.F.) Method," Intern. J. Computer Math., 11, pp. 49-72.

Wing, O., and J. W. Huang, 1980-1, "A Computation Model for Parallel Solution of Linear Equations," IEEE Trans. Comput., C-29, pp. 632-638.

Zaki, H. A., 1986-1, "A Parallelization of the Simplex Method Using the Quadrant Interlocking Factorization," unpublished dissertation, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas.

APPENDIX E

MINIMAL SPANNING TREES: A COMPUTATIONAL INVESTIGATION OF PARALLEL ALGORITHMS

R. S. Barr

R. V. Helgason

J. L. Kennington

Department of Operations Research
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275

PREFACE

The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees. Computational tests were run on a single processor using Prim's, Kruskal's and Boruvka's algorithms. Our implementation of Prim's algorithm is superior for high density graphs, while our implementation of Boruvka's algorithm is best for sparse graphs. Implementations of parallel versions of both Prim's and Boruvka's algorithms were tested on a twenty-cpu Balance 21000. For the environment in which a minimum spanning tree problem is a subproblem within another algorithm, the parallel implementation of Boruvka's algorithm produced speedups of three and five on five and ten processors, respectively; while the parallel implementation of Prim's algorithm produced speedups of three and five on five and ten processors, respectively. The one-time overhead for process creation negates most, if not all of the benefits for solving a single minimum spanning tree subproblem.

ACKNOWLEDGEMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C-0182, the Air Force Office of Scientific Research under Contract Numbers AFOSR 83-0278 and AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and Rome Air Development Center under Contract Number SCEEE PDP/86-75. The authors wish to express their appreciation to Professor Hossam Zaki of the University of Illinois and Professor Iqbal Ali of the University of Massachusetts at Amherst for their helpful comments.

I. INTRODUCTION

The United States along with other developed countries is entering a new generation of computing that will require software engineers to redesign and reevaluate standard algorithms for the new parallel processing hardware that is being installed throughout the developed world. It may well be that algorithms which proved to be superior for single processor machines may prove to be inferior in some of the new parallel processing environments. One of the more popular new parallel machines is Sequent Computer Systems' Balance 21000. The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees on a twenty-cpu Balance 21000.

An _undirected graph_ $G = [V,E]$ consists of a _vertex set_ V and an _edge set_ E. Without loss of generality we assume that the edges are distinct. If $G' = [V',E']$ is a subgraph of G with $V' = V$, then G' is called a _spanning subgraph_ for G. If, in addition, G' is a tree, then G' is called a _spanning tree_ for G. A graph whose components are trees is called a _forest_, and a spanning subgraph for G, which is also a forest, is called a _spanning forest_ for G. We will call $\{[V_i,T_i]: V_i = \{u_i\}, T_i = \emptyset, u_i \in V\}$ the _trivial_ spanning forest for G and the $[V_i,T_i]$ _trivial_ trees. Associated with each edge (u,v) is a real-valued cost $c(u,v)$. The _minimum spanning tree problem_ may be stated as follows: Given a connected undirected graph each of whose edges has a real-valued cost, find a spanning tree of the graph whose total edge cost is minimum.

Applications include the design of a distribution network in which the nodes represent cities or towns and the edges represent electrical power lines, water lines, natural gas lines, communication links, etc. The objective is to design a network which uses the least length of cable or pipe. The minimum spanning tree problem is also used as a subproblem for algorithms for the travelling salesman problem (see Held and Karp [6, 7] and Ali and Kennington [3]). Some vehicle routing algorithms require the solution of a travelling salesman problem on a subset of nodes. Hence, a wide variety of applications require the solution of minimal spanning trees. Some applications require a single solution and some use the model as a subproblem within another algorithm.

II. THREE CLASSICAL ALGORITHMS

The algorithms in current use may be traced to ideas developed by Prim, Kruskal, and Boruvka. These three classical algorithms all begin with the trivial spanning forest $G_0 = \{[V_i,T_i], i = 0,\ldots,|V|-1\}$. A sequence of spanning forests is obtained by merging spanning forest components. Given spanning forest G_k, a nonforest edge (u,v) is selected and the components $[V_i,T_i]$ and $[V_j,T_j]$ with $u \epsilon V_i$ and $v \epsilon V_j$ are removed from G_k and replaced by $[V_\ell,T_\ell]$, where $\ell = k + |V|$, $V_\ell = V_i \cup V_j$, and $T_\ell = T_i \cup T_j \cup \{(u,v)\}$, yielding spanning forest G_{k+1}. After $m = |V|-1$ edges have been selected, $G_m = \{[V_{2m},T_{2m}]\} = \{[V,T]\}$ is a minimal spanning tree for G.

Let $[V_i,T_i]$ and $[V_j,T_j]$ denote two disjoint subtrees of G. Define d_{ij}, the shortest distance between the trees, by $d_{ij} = \min\ \{c(u,v)\colon (u,v) \epsilon E,\ u \epsilon V_i,\ v \epsilon V_j\}$. The three classical algorithms may be viewed as different applications of the following result:

Proposition 1.

Let $V_0, V_1,\ldots,V_n$ denote vertex sets of disjoint subtrees of a minimum spanning tree for G. Let $c(u,v) = d_{jn} = \min\limits_{j\neq n} d_{jn}$ with $(u,v) \epsilon V_j \times V_n$. Then (u,v) is an edge in a minimal spanning tree for G.

A proof of Proposition 1 may be found in Christofides [4, pp. 135-136].

In Prim's algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \epsilon V_i \times V_{j*}$, where $j*$ is the largest index j such that $[V_j,T_j] \epsilon G_k$. Thus a single component continues to grow as trivial trees disappear. An excellent description of Prim's algorithm is given in Papadimitriou and Steiglitz [15, p. 273], along with its (serial) computational complexity of $O(|V|^2)$. It is believed that this algorithm is best suited for dense graphs.

In Boruvka's algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \in V_{i*} \times V_j$, where i* is the smallest index i such that $[V_i,T_i] \in G_k$. Thus a variety of different-sized components may be produced as the algorithm proceeds. All trivial trees will be removed first in the early stages of this algorithm. A description of Boruvka's algorithm is given in Papadimitriou and Steiglitz [15, p. 277], along with its (serial) computational complexity of $O(|E| \log |V|)$. This algorithm appears to be best suited for sparse graphs.

Kruskal's method may be viewed as an application of the greedy algorithm. The minimum spanning tree is constructed by examining the edges in order of increasing cost. If an edge forms a cycle within a component of G_k, it is discarded. Otherwise it is selected and yields G_{k+1}. Here also different-sized components may be produced. A description of Kruskal's algorithm is given in Sedgewick [18, pp. 412-413], along with its (serial) computational complexity of $O(|E| \log |E|)$.

III. COMPUTATIONAL RESULTS WITH SEQUENTIAL ALGORITHMS

Computer codes for Boruvka's algorithm, Kruskal's algorithm, and three versions of Prim's algorithm were developed. SPARSE PRIM maintains the edge data in both forward and backward star format, while DENSE PRIM maintains the edge data in an $|V| \times |V|$ matrix. HEAP PRIM maintains the edge data in both forward and backward star format and makes use of a d-heap as described in Tarjan [19, p. 77]. KRUSKAL makes use of a partial quick sort as described in [1, 8] to produce the least cost remaining edge. BORUVKA is a straightforward implementation of the algorithm presented in [15].

Random problems were generated on both n x n grid graphs and on completely random graphs. All costs were uniformly distributed on the interval [0, maxcost]. All codes are written in FORTRAN for the Balance 21000.

The computational results for grid graphs are presented in Table 1. These graphs are very sparse and BORUVKA was the clear winner. The computational results for random graphs may be found in Tables 2 and 3. SPARSE PRIM was the winner for problems whose density was at least 40% with HEAP PRIM running a close second. For problems with densities of 20% or less, HEAP PRIM was the winner with KRUSKAL running a close second. KRUSKAL appeared to be the most robust implementation, working fairly well on all problems tested.

Table 1. Comparison of Sequential Algorithms on Grid Graphs

(cost range is 0 - 10,000)

Grid Size n x n	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
15 x 15	420	1.7%	1.70	.36	.27	.19	.12
18 x 18	612	1.2%	3.54	.74	.42	.30	.17
20 x 20	760	1.0%	5.43	1.10	.54	.39	.21
24 x 24	1,104	.7%	11.32	2.19	.82	.63	.30
28 x 28	1,512	.5%	21.01	4.09	1.13	.86	.46
30 x 30	1,740	.4%	27.82	5.41	1.37	1.15	.55
Total Time (secs.)			70.82	13.89	4.55	3.52	1.81
Rank			5	4	3	2	1

Table 2. Comparison of Sequential Algorithms on High Density Random Graphs.

(cost range is 0 - 10,000)

Vertices	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
200	19,900	100%	1.39	1.14	1.44	1.52	3.01
200	15,920	80%	1.39	.97	1.22	1.52	1.96
200	11,940	60%	1.39	.79	.99	.96	1.47
200	7,960	40%	1.39	.61	.76	.89	1.02
400	79,800	100%	5.67	4.55	5.42	4.45	12.03
400	63,840	80%	5.69	3.85	4.53	3.58	10.28
400	47,880	60%	5.70	3.13	3.62	2.82	7.26
400	31,920	40%	5.71	2.49	2.68	1.97	4.85
600	179,700	100%	13.28	10.39	11.98	12.38	29.85
600	143,760	80%	13.66	8.79	9.99	14.99	23.72
600	107,820	60%	13.16	7.15	7.99	10.63	17.79
600	71,880	40%	13.02	5.55	5.67	6.05	11.80
Total Time (secs.)			81.45	49.41	56.29	61.76	125.04
Rank			4	1	2	3	5

Table 3. Comparison of Sequential Algorithms on Low Density Random Graphs.

(cost range is 0 - 10,000)

Vertices	Edges	Graph Density	DENSE PRIM	SPARSE PRIM	HEAP PRIM	KRUSKAL	BORUVKA
200	3,980	20%	1.40	.44	.49	.50	.52
200	1,990	10%	1.40	.36	.39	.40	.35
200	995	5%	1.39	.32	.32	.35	.17
400	15,960	20%	5.66	1.75	1.62	1.47	2.46
400	7,980	10%	5.71	1.40	1.12	1.53	1.30
400	3,990	5%	5.72	1.21	.86	1.20	.72
600	35,940	20%	13.04	3.94	3.39	3.99	6.02
600	17,970	10%	13.04	3.05	2.14	2.89	2.86
600	8,985	5%	13.07	2.73	1.50	2.12	1.52
Total Time (secs.)			60.43	15.20	11.83	14.45	15.92
Rank			5	3	1	2	4

IV. PARALLEL ALGORITHMS

Parallel versions of the three classical algorithms have appeared in the literature (see [2, 5, 9, 10, 11, 12, 16, 17]), however; no computation experience has been reported. The overhead required for coordinating the work of multiple processors can only be determined by actual implementation on a parallel processing machine.

A parallel version of Boruvka's algorithm was developed for grid graphs and a parallel version of Prim's algorithm was developed for high density random graphs. Both algorithms use modules (subroutines) which may be executed in parallel. Suppose there are p processors available for use. The parallel operations are initiated by the main program using statements of the form:

for m = 1 to p, <u>fork</u> module <u>z</u>(m).

The main program and p-1 clones will each execute module <u>z</u> in parallel. Processing does not continue in the main program until all processors complete module <u>z</u>. The argument "m" allows each of the p processors to process different parts of the data or follow a different path. We assume that all data in the main program is shared with module <u>z</u>. If module <u>z</u> has local non-shared variables, then these will be explicitly stated in the description of the module. Multiple processors which update the same variable, set, or list use locks to insure that only one processor has access to a given item.

4.1 Parallel Boruvka For Grids

Using the fork and lock constructs we present a parallelization of Boruvka's algorithm for grid graphs. The most expensive component of Boruvka's sequential algorithm may be described by the following procedure:

```
for all $(u,v) \in E$
  let i and j denote the subtrees containing u and v, respectively;
  if $i \neq j$ then
    if $cost(u,v) < min(i)$ then $min(i) \leftarrow cost(u,v)$
    if $cost(u,v) < min(j)$ then $min(j) \leftarrow cost(u,v)$
  end if
end for
```

That is, all the edge costs must be examined and certain subtree data are updated. Our parallelization of this scan relies upon a partitioning of the grid into p components (one for each processor). A three processor partitioning of a 7 x 7 grid network is illustrated in Figure 1.

The above edge scan is performed in two stages. The first stage performs a parallel scan over edges both of whose vertices lie within the same partition. The second stage performs a parallel scan over edges across cut sets. If each partition consists of at least two rows of the grid, then all subtree data updating can be performed independently without the requirement of a lock.

The second part of Boruvka's algorithm is to merge two subtrees by appending a new edge. The merger of subtrees, both of which lie in the same partition can also be executed in parallel.

Using this data partitioning approach, the parallel algorithm may be stated as follows:

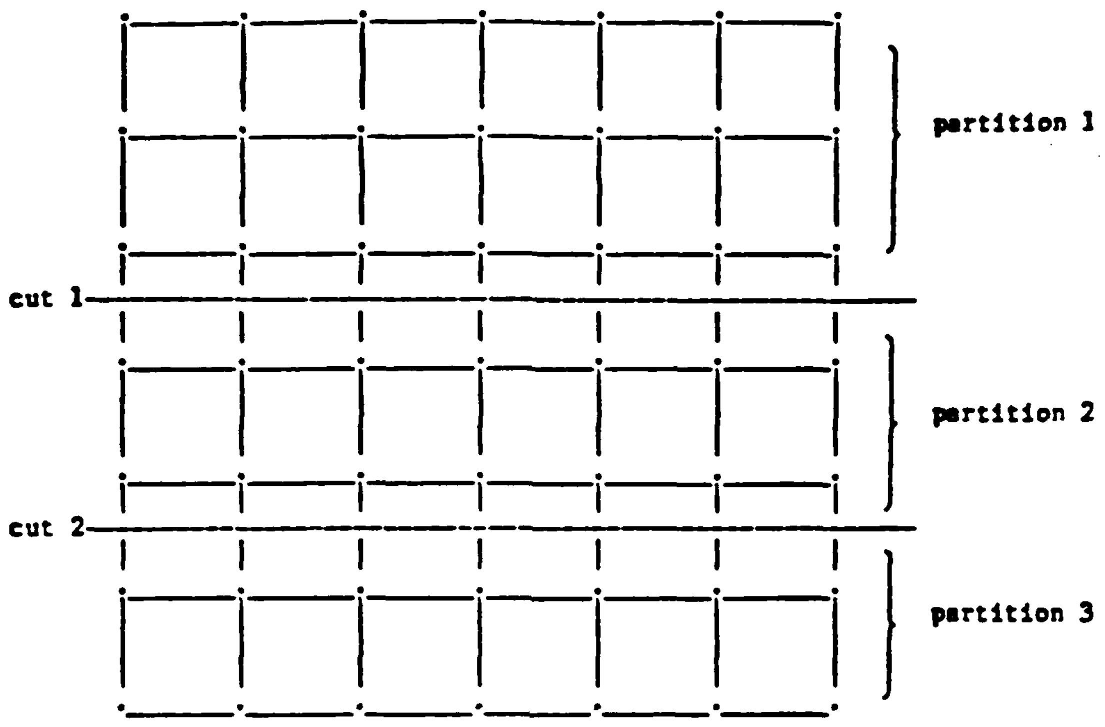


Figure 1. A Three Processor Partitioning of a 7 x 7 Grid Graph.

PARALLEL BORUVKA FOR GRIDS

```
Input:       1. An n x n grid graph $G = [V,E]$ with $V = \{v_1,\ldots, v_q\}$.
             2. For each edge $(u,v) \in E$ a cost $c(u,v)$.
             3. The number of processors, p, available for use.
Output:      A minimal spanning tree $[V,T]$.
Assumption:  G is connected and has no parallel edges.
begin
    $T \leftarrow \emptyset$, $r \leftarrow \lceil n/p \rceil$, $\ell \leftarrow n - rp$;
    If $r < 2$, terminate.
    for $i = 1$ to q,  $S_i \leftarrow \{v_i\}$;
    $C \leftarrow \{S_1,\ldots, S_q\}$;
    $W_1 \leftarrow \{v: v \in V$ and v is in grid rows 1 through $v + \ell\}$;
    for $m = 2$ to p,
        $W_m \leftarrow \{v: v \in V$ and v is in grid rows $(m-1)r + \ell + 1$ through $mr + \ell\}$;
    for $m = 1$ to p, $X_{1m} \leftarrow \{(u,v): (u,v) \in E$, $u \in W_m$, and $v \in W_m\}$;
    for $m = 1$ to $p - 1$,
        $X_{2m} \leftarrow \{(u,v): (u,v) \in E$ with $u \in W_m$, $v \in W_{m+1}$ or $u \in W_{m+1}$, $v \in W_m\}$;
    for $i = 1$ to q, cpu(i) $\leftarrow$ m, where $v_i \in W_m$;
    (comment: $S_1,\ldots, S_q$ are assigned to the p processes)
    create p-1 clones
    (comment: create p-1 additional processes and place them in the wait
              state)
    while $|C| \neq 1$
        for $m = 1$ to p, fork module edgescan(1,m);
        (comment: forks are executed in parallel and processing does not continue
                  in the main program until all processes complete edgescan)
        for $m = 1$ to p-1, fork module edgescan(2,m);
        $L \leftarrow \emptyset$;
```

```
            for m = 1 to p, fork module merge(m);
            for all (u,v) ε L do
               let $S_i$ and $S_j$ be the sets containing u and v, respectively;
               if $|S_i| < |S_j|$ then
                   $S_i \leftarrow S_i \cup S_j$, $C \leftarrow C \setminus S_j$;
               else
                   $S_j \leftarrow S_i \cup S_j$, $C \leftarrow C \setminus S_i$;
               end if
               $T \leftarrow T \cup (u,v)$;
            end for
         end while
         kill the clones
     end
     module edgescan(k,m)
     begin
     (comment: k = 1 implies the scan is within partition m,
               k = 2 implies the scan is across the cut set separating partitions
                     m and m + 1)
            for all (u,v) ε $X_{km}$
              let $S_i$, $S_j$ be the sets containing u and v, respectively;
              if $i \neq j$ then
                if c(u,v) < min(i) then min(i) ← c(u,v), shortest(i) ← (u,v);
                if c(u,v) < min(j) then min(j) ← c(u,v), shortest(j) ← (u,v);
              end if
              (comment: shortest(i) is the least cost edge incident on $S_i$)
            end for
     end
```

```
module merge(m)
begin
       for all $v_k \in W_m$ do
         (u,v) ← shortest(k)
         let $S_i$, $S_j$ be the sets containing u and v, respectively;
         if $i \neq j$ then
           if cpu(i) = cpu(j) then
             if $|S_i| < |S_j|$ then
               $S_i \leftarrow S_i \cup S_j$, $C \leftarrow C \backslash S_j$;
             else
               $S_j \leftarrow S_i \cup S_j$, $C \leftarrow C \backslash S_i$;
             end if
             lock T
               $T \leftarrow T \cup \{(u,v)\}$
             unlock T
           else
             lock L
               $L \leftarrow L \cup \{(u,v)\}$
             unlock L
           end if
         end if
       end for
end
```

4.2 Parallel Prim

The most expensive part of Prim's sequential algorithm is to find a minimum entry in an $|V|$ length array. This search can be allocated over p processors, each of which finds a candidate minimum. The best of the p candidates becomes the global minimum. Under the assumption that parallel edges do not exist, there is also a scan of edges over the forward and backward star of a given node which can be executed in parallel. Data partitioning via the use of independent cut sets could also be used for random graphs in a manner similar to that described in Section 4.1. That has not been done in this investigation.

The parallelization of Prim's algorithm may be stated as follows:

```
                              PARALLEL PRIM

Input:      1. A graph $G = [V,E]$ with $V = \{v_1, \ldots, v_n\}$.
            2. For each edge $(u,v) \in E$, a cost $c(u,v)$.
            3. The number of processors, p, available for use.
Output:     A minimal spanning tree, [V,T].
Assumption: G is connected and has no parallel edges.
begin
    $U \leftarrow \{v_1\}$, $w \leftarrow v_1$, $T \leftarrow \emptyset$ ;
    for $i = 1$ to n, $d(i) \leftarrow \infty$ ;
    create p-1 clones
    (comment: create p-1 additional processes and place them in a wait
              state)
    $F \leftarrow \{(w,v) \in E\}$;
    partition F into mutually exclusive sets $F_1, \ldots, F_s$, $s \leq p$;
    for $m = 1$ to s, fork module forwardscan(m);
    $B \leftarrow \{(u,w) \in E\}$;
```

```
partition B into mutually exclusive sets $B_1,\ldots,B_t$, $t \leq p$;
for m = 1 to t, fork module backwardscan(m);
  while $U \neq V$ do
    globalmin $\leftarrow \infty$;
    for m = 1 to p, fork module nodescan(m);
    (comment: forks are executed in parallel and processing does not
              continue in the main program until all processes complete
              nodescan)
    $T \leftarrow T \cup \{e(ibest)\}$, $U \leftarrow U \cup \{w\}$;
    $F \leftarrow \{(w,v) \in E\}$;
    partition F into mutually exclusive sets $F_1,\ldots, F_s$, $s \leq p$;
    for m = 1 to s, fork module forwardscan(m);
    $B \leftarrow \{(u,w) \in E\}$;
    partition B into mutually exclusive sets $B_1,\ldots, B_t$, $t \leq p$;
    for m = 1 to t, fork module backwardscan(m);
  end while
  kill the clones
end
```

```
module nodescan(m)
local data: min, x
begin
  min ← ∞
  for i = m to n step p do
    if d(i) < min then min ← d(i), x ← i;
  end for
  lock globalmin
    if min < globalmin then globalmin ← min, ibest ← x, w ← $v_x$;
  unlock globalmin
end

  module fowardscan(m)
  begin
    for all (u,v) ε $F_m$ do;
      if c(u,v) < d(v) then d(v) ← c(u,v), e(v) ← (u,v);
    end for
  end
  module backwardscan(m)
  begin
    for all (u,v) ε $B_m$ do;
      if c(u,v) < d(u) then d(u) ← c(u,v), e(u) ← (u,v);
    end for
  end
```

V. COMPUTATIONAL RESULTS WITH PARALLEL ALGORITHMS

Both algorithms of Section IV were coded in FORTRAN for the Balance 21000 located in the Center for Applied Parallel Processing at Southern Methodist University. The Balance 21000 is configured with twenty NS32032 cpu's, 32 Mbytes of shared memory, and 16K user-accessible hardware locks. Each cpu has 8 Kbytes of local RAM and 8 Kbytes of cache. The Balance 21000 runs the DYNIX operating system, a version of UNIX 4.2bsd. DYNIX includes routines to create, synchronize, and terminate parallel processes from C, Pascal, and FORTRAN. More details about the Balance 21000 may be found in [13].

Table 4 gives the computational results with Boruvka's algorithm. The times are wall clock times and are the average for three runs. The first row in each table contains the time for the sequential version of BORUVKA and all other rows contain times for the parallel version. The sequential version is 250 lines of code, while the parallel version required over 400 lines. The speedup for a row is calculated by dividing the best sequential time by the time in that row.

Initially, the parallel code creates the additional processes to be used and requires each of them to build data tables which give the location in virtual memory of all shared data. Once this is done, the processes can be used repeatedly with little system overhead. However, this initial creation and the subsequent killing of those processes at termination can be very expensive for this type of problem. The first column of times includes the creation and process termination time while the second does not. Hence, if a 350 x 350 minimal spanning tree was to be obtained one time, then the best speedup is 2.6 using seven cpu's. If however, this is a subprogram of a larger system, then a 350 x 350 problem can yield a speedup of four on six processors and a speedup of five on ten.

Table 4. Parallel Boruvka on 350 x 350 Grid Graph
|V| = 122,500 |E| = 244,300
(cost range is 0 - 100,000)

cpu's	PARALLEL BORUVKA (includes process creation)		PARALLEL BORUVKA (excludes process creation)	
	time	speedup	time	speedup
1+	98.21	1.00	98.21	1.00
1*	112.57	.87	103.86	.95
2	66.93	1.47	57.49	1.71
3	50.26	1.95	40.92	2.40
4	40.25	2.44	29.95	3.28
5	39.00	2.52	26.52	3.70
6	38.69	2.54	23.45	4.19
7	37.70	2.60	21.62	4.54
8	40.98	2.40	21.58	4.55
9	42.49	2.31	20.85	4.71
10	41.30	2.38	17.52	5.61

+ best sequential BORUVKA code
* parallel code run with a single processor

Table 5. Parallel Prim on G = [V,E] with |V| = 900 and |E| = 404,550

(cost range is 0 - 100,000)

cpu's	PARALLEL PRIM (includes process creation)		PARALLEL PRIM (excludes process creation)	
	time	speedup	time	speedup
1+	24.88	1.00	24.88	1.00
1*	27.09	.92	26.98	.92
2	23.35	1.07	15.12	1.65
3	22.63	1.10	10.84	2.30
4	25.31	.98	8.74	2.85
5	28.43	.88	7.39	3.37
6	31.54	.79	6.62	3.76
7	36.51	.68	6.03	4.13
8	41.08	.61	5.62	4.43
9	46.04	.54	5.30	4.69
10	50.54	.49	5.02	4.96

+ best sequential PARALLEL PRIM code
* parallel code run with a single processor

Table 5 gives the computational results with Prim's algorithm. No speedup is achievable for a one-time solution. For environments in which the minimum spanning tree problem is a subproblem, speedups of <u>three</u> and <u>five</u> were obtained on five and ten processors, respectively.

VI. SUMMARY AND CONCLUSIONS

Five computer codes were developed to solve the minimum spanning tree problem on a sequential machine. These codes were computationally compared on both grid graphs and random graphs whose densities varied from 5% to 100%. The implementation of Boruvka's algorithm (see [15, p. 277]) was the best for grid graphs. An implementation of Prim's algorithms using a sparse data representation (see [15, p. 273]) was best for high density random graphs while an implementation of Prim's algorithm using a d-heap (see [19, p. 77]) was best for lower density random problems. Kruskal's algorithm using a quicksort is the most robust of all the implementations, ranking either second or third in all computational tests. Both Boruvka's and Prim's algorithms were parallelized by the method of data partitioning (also called homogeneous multitasking). This involves creating multiple, identical processes and assigning a portion of the data to each processor. For the environment in which a minimal spanning tree problem is a subproblem within a larger system, speedups of <u>five</u> on ten processors were achieved with both Prim's and Boruvka's algorithms. The overhead for parallel processing on the Balance 21000 negates most of the benefits of parallel processing for the first solution of the minimal spanning tree.

REFERENCES

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts (1974).

2. Akl, S., "An Adaptive and Cost-Optimal Parallel Algorithm for Minimum Spanning Trees," Computing, 36 (1986) 271-277.

3. Ali, I., and J. Kennington, "The Asymmetric M-Travelling Salesman Problem: A Duality Based Branch-And-Bound Algorithm," Discrete Applied Mathematics, 13 (1986) 259-276.

4. Christofides, N., Graph Theory: An Algorithmic Approach, Academic Press, New York, New York (1975).

5. Deo, N., and Y. Yoo, "Parallel Algorithms for the Minimum Spanning Tree Problem," Proceedings of the 1981 International Conference on Parallel Processing, IEEE Computing Society Press, (1981) 188-189.

6. Held, M., and R. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees," Operations Research, 18 (1970) 1138-1162.

7. Held, M., and R. Karp, "The Travelling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming, 1 (1970) 6-25.

8. Knuth, D. E., Sorting and Searching, Addison-Wesley, Reading, Massachusetts (1973).

9. Kwan, S., amd W. Ruzzo, "Adaptive Parallel Algorithms for Finding Minimum Spanning Trees," Proceedings of the 1984 International Conference on Parallel Processing, IEEE Computing Society Press, (1984) 439-443.

10. Lavallee, I., and G. Roucairol, "A Fully Distributed (Minimal) Spanning Tree Algorithm," Information Processing Letters, 23 (1986) 55-62.

11. Lavallee, I., "An Efficient Parallel Algorithm for Computing a Minimum Spanning Tree," Parallel Computing 83, (1984) 259-262.

12. Nath, D., and S. Maheshwari, "Parallel Algorithms for the Connected Components and Minimal Spanning Tree Problems," Information Processing Letters, 14, 1 (1982) 7-11.

13. Osterhaug, A., Guide to Parallel Programming on Sequent Computer Systems, Sequent Computer Systems, Inc., Beaverton, Oregon (1986).

14. Parallel Computers and Computations, Editors J. van Leevwen and J. K. Lenstra, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, (1985).

15. Papadimitriou, C. and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey (1982).

16. Pawagi, S. and I. Ramakrishnan, "An O(log n) Algorithm for Parallel Update of Minimum Spanning Trees," Information Processing Letters, 22 (1986) 223-229.

17. Quinn, M. J., Designing Efficient Algorithms for Parallel Computers, McGraw-Hill, New York, New York (1987).

18. Sedgenwick, R., Algorithms, Addison-Wesley, Reading, Massachusetts (1983).

19. Tarjan, R. E., Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1983).

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.