Very Large-Scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods

Robert Bixby *John Gregory*
Irvin Lustig *Roy Marsten*
David Shanno

CRPC-TR91152
May 1991

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Appears in Operations Research, Vol. 40, No. 5, September-October 1992.

VERY LARGE-SCALE LINEAR PROGRAMMING: A CASE STUDY IN COMBINING INTERIOR POINT AND SIMPLEX METHODS

ROBERT E. BIXBY
Rice University, Houston, Texas

JOHN W. GREGORY
Cray Research, Inc., Eagan, Minnesota

IRVIN J. LUSTIG
Princeton University, Princeton, New Jersey

ROY E. MARSTEN
Georgia Institute of Technology, Atlanta, Georgia

DAVID F. SHANNO
Rutgers University, New Brunswick, New Jersey

(Received May 1991; revision received January 1992; accepted February 1992)

Experience with solving a 12,753,313 variable linear program is described. This problem is the linear programming relaxation of a set partitioning problem arising from an airline crew scheduling application. A scheme is described that requires successive solutions of small subproblems, yielding a procedure that has little growth in solution time in terms of the number of variables. Experience using the simplex method as implemented in **CPLEX**, an interior point method as implemented in **OB1**, and a hybrid interior point/simplex approach is reported. The resulting procedure illustrates the power of an interior point/simplex combination for solving very large-scale linear programs.

Recent improvements in implementations of the simplex method as well as developments in interior point methods have changed our concept of large-scale linear programming. In this study, experience in solving the linear programming relaxation of a large set partitioning problem on a CRAY Y-MP (CRAY and CRAY Y-MP are trademarks of Cray Research, Inc.) supercomputer is reported. The linear program has 837 rows with 12,753,313 columns and 99,845,826 nonzeros and arises from airline crew scheduling. Using a number of techniques, the solution time is reduced to approximately 4 minutes with a 3.5 minute preprocessing time.

The basic procedure (called *sifting*) is a kind of column generation method. Its application for airline crew scheduling was proposed by John Forrest (1989), but in principle, it can be applied to many problem classes. Computational experience is reported for this procedure using two different methods to solve the linear programs that are generated: **CPLEX**, (a trademark of CPLEX Optimization Inc.) an implementation of the simplex method by Bixby (1990), and **OB1**, an implementation of the primal-dual predictor-corrector interior point method by Lustig, Marsten and Shanno (1990a). In addition, a hybrid interior point-simplex implementation is explored.

In Section 1, the characteristics of the set partitioning problems are discussed and in Section 2, the sifting procedure is described. Section 2.2 describes a new pricing rule. Sections 3 and 4 describe our experience with the sifting procedure using the simplex method and an interior point method, respectively. This experience led to the development of a hybrid scheme, using both the interior point and the simplex methods. It is described in Section 5. This hybrid scheme illustrates how interior point and simplex methods can be combined effectively to exploit the relative advantages of each method.

Subject classification: Programming, linear: large-scale systems.
Area of review: COMPUTING.

0030-364X/92/4005-0885 $01.25

1. A SET PARTITIONING PROBLEM

Set partitioning models have the form (Nemhauser and Wolsey 1988):

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = e, \\ & x \geq 0, \\ & x \quad \text{integer} \end{aligned} \tag{1}$$

where $A \in \{0, 1\}^{m \times n}$, $e = (1, 1, \ldots, 1)^T \in \Re^m$, and $c \in \Re^n$. In the problems studied, $c > 0$. The impetus for this work was the "American Airlines Challenge." American Airlines uses a set partitioning model of the flight crew scheduling problem (Gershkoff 1989). (For a recent discussion of crew scheduling, see Barutt and Hull 1990.) To stimulate fresh thinking on the solution of such models, American generated an instance with 837 rows and approximately 12,750,000 columns. They then made the data available to researchers. The ultimate challenge is to obtain an optimal integer solution. As a step toward that goal, the sifting procedure was developed and used to solve the associated linear programming relaxation (obtained by dropping integrality in (1)).

Airline crew scheduling problems exhibit a number of interesting characteristics. Due to the way the columns are generated (Gershkoff), the matrix A contains many duplicate columns. Duplicate columns are columns that have nonzeros in exactly the same rows, but possibly different cost coefficients. Eliminating all but one of these columns can significantly reduce the size of the problem without affecting the optimal solution.

In typical airline crew scheduling problems there are an average of about 10 ones per column, with a maximum of 18 in the instance provided by American. The cost coefficients c are highly variable (with many integer values in the range from 0 to 10,000 in the instance provided). On the other hand, the right-hand side is constant. Thus, (1) is typically highly degenerate, but its dual is not. This fact plays a significant role in the solution strategies used.

Primarily because of degeneracy, but also because of slow convergence even when objective function progress is being made, relaxations of set partitioning problems have long been considered difficult. Even small numbers of columns can result in a problem that is troublesome for standard implementations of the primal simplex method. Recent developments of dual simplex and interior point methods have largely eliminated that difficulty, at least for moderate sized instances. As Table I shows, these approaches achieve substantial improvements over even a rather strong version of primal simplex using steepest-edge pricing. However, the growth of solution times in Table I with problem size also demonstrates quite clearly that direct application of even these latter approaches is not sufficient to deal with the full 12.75 million variable problem. This is especially true since solving the integer program will require multiple solutions of the linear programming relaxation.

Fortunately, set partitioning models have an important property that allows their linear programming relaxations to be solved by nondirect methods. Let $\mathscr{W} \subset \{1, \ldots, n\}$ be a subset of columns such that the

Table I
Direct Solution of Problems With **CPLEX** and **OB1**
Where SE is Steepest Edge[a]

Number of Columns $\hat{n}$	Nonduplicate Columns	Method	Iterations	CRAY CPU Time (Secs.)
25,000	17,937	**CPLEX** Dual SE	302	1.4
25,000	17,937	**CPLEX** Primal SE	1,768	22.6
25,000	17,937	**OB1**	16	17.3
50,000	35,331	**CPLEX** Dual SE	509	4.8
50,000	35,331	**CPLEX** Primal SE	8,233	179.7
50,000	35,331	**OB1**	18	22.9
100,000	68,428	**CPLEX** Dual SE	1,215	27.3
100,000	68,428	**OB1**	21	34.1
200,000	134,556	**CPLEX** Dual SE	2,549	116.6
200,000	134,556	**OB1**	30	67.7

[a] Problems are generated by taking the first $\hat{n}$ columns from the input file, where $\hat{n}$ = (25,000, 50,000, 100,000, 200,000).

linear program

$$\begin{aligned} &\text{minimize} && c_{\mathscr{W}}^T x_{\mathscr{W}} \\ &\text{subject to} && A_{\mathscr{W}} x_{\mathscr{W}} = e, \\ & && x_{\mathscr{W}} \geq 0, \end{aligned} \tag{2}$$

is feasible. Then this problem can be considered as a smaller instance of (1) because columns in $A_{\mathscr{W}}$ have the same structure as *all* columns in A. It is this property along with the fact that n is *significantly larger* than m that makes "sifting" natural for (1), and also makes sifting natural for a number of other problems, such as the traveling salesman problem and path formulations of appropriate multicommodity flow problems.

2. SIFTING

As noted in the Introduction, the sifting procedure applied here was suggested by Forrest, who termed it *sprint*. Consider the general linear program

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \\ & && x \geq 0, \end{aligned} \tag{3}$$

with $A \in \Re^{m \times n}$, $c \in \Re^n$, $b \in \Re^m$. The dual of (3) is

$$\begin{aligned} &\text{maximize} && b^T y \\ &\text{subject to} && A^T y \leq c. \end{aligned} \tag{4}$$

The sifting procedure begins by taking a "working set" of columns $\mathscr{W} \subset \{1, \ldots, n\}$ such that

$$\begin{aligned} &\text{minimize} && c_{\mathscr{W}}^T x_{\mathscr{W}} \\ &\text{subject to} && A_{\mathscr{W}} x_{\mathscr{W}} = b, \\ & && x_{\mathscr{W}} \geq 0, \end{aligned} \tag{5}$$

is feasible. (This assumption is not essential.) Let π^* be an optimal solution to

$$\begin{aligned} &\text{maximize} && b^T \pi \\ &\text{subject to} && A_{\mathscr{W}}^T \pi \leq c_{\mathscr{W}}, \end{aligned} \tag{6}$$

the dual of (5), and let $x_{\mathscr{W}}^*$ be an optimal solution of (5). Then the vector $x^T = ((x_{\mathscr{W}}^*)^T, 0) \in \Re^n$ is optimal for (3) if

$$c - A^T \pi^* \geq 0. \tag{7}$$

The sifting procedure suggested by these observations may be summarized as follows:

Given the linear program (3) and a set $\mathscr{W}$ such that (5) is feasible:

Solve (5) obtaining x^* and π^*.

while $(c - A^T \pi^* \not\geq 0)$ **do** (major iteration)
  Choose $\mathscr{P} \subset \{1, \ldots, n\} \backslash \mathscr{W}$. (price)
  Set $\mathscr{W} \leftarrow \mathscr{W} \cup \mathscr{P}$. (augment problem)
  (Optionally) If $\mathscr{W}$ is too big, reduce the size of $\mathscr{W}$. (purge)
  Solve (5) obtaining x^* and π^*. (solve)
end while.

In the following sections, the key ideas in implementing the above procedure are discussed:

1. The efficient computation of $A^T \pi^*$ given π^* is discussed in Section 2.1.
2. Choosing a good set of candidates for $\mathscr{P}$ is discussed in Section 2.2, where a new method for normalizing reduced costs is introduced.
3. Controling the size of $\mathscr{W}$ via control of the size of $\mathscr{P}$ and purges of $\mathscr{W}$ is discussed in Section 2.3.
4. Finally, the specific algorithms used to solve (5) are treated. That discussion is broken into three parts, the simplex method (Section 3), an interior point method (Section 4), and a hybrid interior point/simplex approach (Section 5). These three sections are preceded by a discussion (Section 2.4) of a number of issues applicable to each of the three solution methods.

2.1. Vectorization of the Pricing Operation

For each major iteration of the sifting procedure, the values $A^T \pi^*$ are computed. However, on a vector supercomputer, the usual direct procedure to compute $A^T \pi^*$ can exhibit poor performance if most of the columns in A are short. To get effective vectorization, the algorithm described by Forrest and Tomlin (1990) is used. For an explanation of vectorization issues as related to optimization, see Zenios and Mulvey (1986).

Let l_j represent the number of nonzeros in column j. Since $l_j \leq m$, a simple bucket sort (using $O(n)$ time) can be used to sort the columns according to increasing column length. Given this ordering, let n_l represent the number of columns with l nonzeros and let j_l represent the first column (in the new ordering) with l nonzeros. Then it follows that columns $j_l, j_l + 1, \ldots, j_l + n_l - 1$ all have the same column length. To compute $d_j = A_j^T \pi^*$, for $j_l \leq j \leq j_l + n_l - 1$ and a fixed value of l, the following procedure is used, where i_{kj} represents the row number of the kth nonzero in

column j:

$d_j = 0$ **for** $j_l \leq j \leq j_l + n_l - 1$
for $k = 1$ **to** l **do**
  for $j = j_l$ **to** $j_l + n_l - 1$ **do**
    $d_j = d_j + \pi^*_{i_{kj}} A_{i_{kj}j}$.

This loop is repeated for each value of l. The loop vectorizes well if the nonzeros of each column and the row indices i_{kj} are stored consecutively. This happens when the values n_l are large as compared to the maximum value of l_j. This procedure was found to be approximately ten times faster than the usual direct procedure for computing d_j by

$$d_j = \sum_{k=1}^{l_j} \pi^*_{i_{kj}} A_{i_{kj}j}. \tag{8}$$

The inner loop of the new procedure is making a calculation with a large vector of average length $n_l/10$, as opposed to the small vectors of average length 10 used in the direct procedure.

2.2. A New Pricing Rule

The initial experiments with solving the full problem used the following procedure that chooses a target number t of columns j with small reduced cost $c_j - A_j^T\pi^*$:

Procedure standard_price

Step 1. Compute $\bar{c} = c - A^T\pi^*$.

Step 2. Find $\mathscr{P} = \{j: \bar{c}_j < 0\}$.

Step 3. If $\mathscr{P}$ has more than t columns, then sort (in ascending order) the values $\bar{c}_j$ for $j \in \mathscr{P}$ to find the t columns to keep in $\mathscr{P}$ with the most negative values of $\bar{c}_j$.

Step 1 was carried out as described in Section 2.1. Step 2 used the CRAY subroutine **WHENFLE** and the sort in Step 3 was done by using the CRAY **ORDERS** routine. The set $\mathscr{P}$ determined by **standard_price** was added to the current working set $\mathscr{W}$. The values t used on each iteration are discussed in Section 2.3.

For the particular set partitioning problems considered, this pricing procedure produced adequate performance. However, in working directly with small versions of these problems, it was observed that the steepest-edge pricing rule (Goldfarb 1976, Goldfarb and Reid 1977) with the simplex method produced much better results than simply using reduced costs. However, the steepest-edge rule is inherently based on the simplex method. Our desire was to use a column selection strategy that is as independent as possible of the algorithm used to solve the subproblem. An analysis of the problem yielded a new pricing rule that performed well for these problems.

Given an optimal solution π^* to the dual (6) of (5), let

$$\lambda = \min_{1 \leq j \leq n} \left\{ \frac{c_j}{A_j^T\pi^*} : A_j^T\pi^* > 0 \right\}. \tag{9}$$

Then, since $c \geq 0$, it is clear that $\lambda \geq 1$ if and only if $\bar{c}_j = c_j - A_j^T\pi^* \geq 0$ for all j. Furthermore, if $A_j^T\pi^* \leq 0$ for all j, then $y = \pi^*$ is optimal for the dual problem (4). It follows from the definition of λ that $A^T(\lambda\pi^*) \leq c$, so that the vector $\lambda\pi^*$ provides a feasible solution $y = \lambda\pi^*$ to

$$\begin{aligned} &\text{maximize} && e^T y \\ &\text{subject to} && A^T y \leq c, \end{aligned} \tag{10}$$

the dual of the linear programming relaxation of (1). In fact, the value $\lambda e^T\pi^*$ provides a lower bound on the objective value for the full primal linear programming relaxation of (1). This bound may be useful in other procedures that use column generation on classes of linear programs with nonnegative costs.

Intuitively, the constraints in (10) yielding the smallest values of $c_j/A_j^T\pi^*$ are most likely to be active at the optimal solution. Furthermore, choosing columns with minimal values of $c_j/A_j^T\pi^*$ is equivalent to choosing columns with minimal values of $\bar{c}_j/c_j$ when $c_j > 0$. Hence, using $c_j/A_j^T\pi^*$ is the same as scaling the reduced costs by the original cost coefficient. This idea suggests the following procedure for selecting a target t number of columns:

Procedure lambda_price

Step 1. Compute $d = A^T\pi^*$.

Step 2. Compute $\bar{c} = c - d$.

Step 3. Find $\mathscr{P} = \{j: \bar{c}_j < 0\}$.

Step 4. If $\mathscr{P}$ has t or less columns, then exit.

Step 5. Compute $\lambda_j = c_j/d_j$ for columns $j \in \mathscr{P}$.

Step 6. Sort (in ascending order) the values λ_j for $j \in \mathscr{P}$.

Step 7. Set $\mathscr{P}$ to the t columns with the lowest values of λ_j.

Note that if $j \in \mathscr{P}$, then $c_j - A_j^T\pi^* < 0$ which implies that $\lambda_j < 1$ and $A_j^T\pi^* > 0$ so that the above steps are well defined. This pricing rule should prefer columns with low absolute cost as well as columns with more

nonzeros. Experimentation with the initial pricing rule indicated that these columns should be preferred as these types of columns are predominant in the optimal solution.

As compared to the **standard_price** procedure, there is a small amount of extra work represented by Step 5 in **lambda_price**. This step vectorizes well and only added a small cost to the pricing procedure. This cost was small compared to the savings generated by the reduction in the number of outer iterations. The new pricing procedure generally seems to pick better columns for the sifting procedure. Table II indicates the number of major iterations of the sifting procedure using the two different pricing procedures. **CPLEX** was used to solve the subproblems. The different problems are defined in Section 2.4.

2.2.1. Degeneracy and Multiple Dual Optimality

It is well known that the linear programming relaxations of set partitioning problems exhibit massive primal degeneracy. For the sifting procedure, the most important implication is the presence of multiple dual optimal solutions. The particular choice of optimal dual solution influences the selection of new columns.

If the subproblem (5) is solved by the simplex method, the optimal basis chosen is essentially random, changing the qualitative nature of an optimal dual solution. If the subproblem (5) is solved by an interior point method, the solution will be in the relative interior of the face of optimal dual solutions. The particular dual solution π^* chosen depends on the initial point used by the solution algorithm.

2.3. Problem Size Control

One of the most difficult decisions in implementing the sifting procedure is control of the cardinality of $\mathscr{W}$. We chose to specify a sequence $(t^1, t^2, \ldots)$ of target values, such that the cardinality of $\mathscr{P}$ is restricted to be smaller than t^k on the kth iteration of the sifting procedure. In addition, it is important to occasionally purge columns from $\mathscr{W}$ as an additional control on the size of the subproblems. The details of these procedures and the specific values of t^k for each of the three solution methods are discussed in the appropriate sections. Unfortunately, one weakness of the algorithm as implemented is that these sequences were determined by experimentation for the specific problem being studied. However, once a reasonable sequence was determined, it was applied *unchanged* to all problem instances. Thus, although the largest problem solved has approximately 12.75 million columns, we would choose exactly the same sequence for a problem of larger size.

It may seem more natural to simply build the subproblems by including in $\mathscr{P}$ all columns with negative reduced cost. That approach is simply impractical. For example, 10% of the columns have negative reduced costs on the first iteration. We experimented with rules for choosing $\mathscr{P}$ on iteration k on the basis of whether $\bar{c}_j < -\delta^k$, where $\delta^k \geq 0$ is large for small values of k and is reduced on subsequent iterations. This type of rule did not effectively control the size of $\mathscr{W}$ on each iteration and was not robust over the problem set. Furthermore, different strategies for controlling the size of $\mathscr{W}$ seemed necessary depending upon the solution method. Experiments with varying the sequence t^k increased the solution times by factors ranging from 1.1 to 2.5 for the largest problem instance. This experimentation led us to a greater understanding of the need for problem size control. For sifting to be applied to other problems, the general concept of problem size control seems to be the dominant factor in the choice of the sequence t^k.

Table II
Major Iterations for Two Pricing Rules

Problem Name	Major Iterations	
	standard_price	**lambda_price**
1mil	12	9
2mil	11	9
4mil	10	11
8mil	15	12
13mil	17	12

2.4. Engineering the Problem

Our experiments were run on one processor of a CRAY Y-MP computer with 128 megawords of memory running UNICOS 6.0.11 with the `cft77` FORTRAN compiler version 5.0.0 and the `scc` C compiler version 3.0.0. At most 28 megawords were required to solve any of the problems. In attempting to solve such a linear program, a number of tasks are required to prepare the problem for solution.

The data were provided in 7 ASCII (machine independent format) files. Each line of each file corresponds to one column in the constraint matrix. A simple preprocessor was used to read in this file in 250,000 column sections and write out 51 binary files consisting of a slightly modified image of the original data. Because of the nature of the ASCII files, 39 sections had exactly 250,000 columns, ten sections had 250,302 columns, one section had 249,998 columns, and one section had 250,297 columns. This

process of converting the data to a binary format is essentially a required input step. Moreover, it is reasonable to assume that the data could be provided in this binary format initially. In fact, the time to read in the 7 ASCII files (approximately 1,400 CRAY Y-MP CPU seconds) exceeds the solution time! The particular format of these binary files stored each section of data as 3 vectors, a cost vector, the vector l of nonzero counts per column, and a vector of the indices of rows with nonzeros. These three vectors are written to each file as separate binary vectors in order to enhance vectorization.

For the problem that was provided, the first 1,000 columns constitute an initial feasible solution. This set of columns was used as the initial $\mathscr{W}$ for all solution methods. The performance of the sifting method on this model is very insensitive to the size of this initial feasible set. The feasibility of the first 1,000 columns is a property of the output of the program used by American to generate the columns. A selection of a different set of initial columns did not seem to alter the overall performance of the algorithm.

To reduce the cost of computing $A^T\pi^*$ on each iteration, duplicate columns were removed from each of the 51 sections. Let $\mathscr{S} \subset \{1, \ldots, n\}$ denote the columns within some particular section. Given $\mathscr{S}$, all of the nonduplicate columns in $\mathscr{S}$ are found before sorting the columns of $\mathscr{S}$ by the number of nonzeros per column, as described in Section 2.1. Note that this step does not remove all nonduplicates from A, as duplicate columns may still exist across section boundaries. Because of this, duplicate columns were eliminated among $\mathscr{W}$ on each major iteration of the sifting procedure before the subproblem (5) was optimized.

For the 12,753,313 column problem provided by American, duplicates were removed from each of the 51 sections, reducing the size of each section to an average size of 167,645 columns. This process consumed 212 CRAY Y-MP CPU seconds, plus an estimated 103 additional overhead seconds for input of the binary data files and output of the binary nonduplicate files, as described below. Table III presents statistics for the sequence of five problems generated from the initial data that were used for the tests in this study and the names we assigned for reference to each problem. The value s is the number of sections used to create the problem instance. Note that the sizes of the problems are not exact multiples of 1,000,000 because of the way the original data sets were provided. In particular, one of the original files contained only 1,999,998 columns, rather than the expected 2,000,000.

The size of each section was chosen large enough to get good vectorization performance during the pricing operation and small enough so as not to consume too much memory when loaded from external storage. Each section of nonduplicate columns is stored in an external file. On the CRAY Y-MP, experiments were conducted with three choices for the locations of the external files. The first choice was a UNICOS 5.0 file system, the second a UNICOS 6.0 file system, and the third a solid-state storage device (SSD) that essentially is a RAM-disk. It was found that using SSD to store the external files produced a negligible overhead for the sifting procedure, and hence, all further results are reported assuming SSD usage. Each section of nonduplicate files is stored as a binary file on the SSD. The values n_l, i_{kj} (Section 2.1) and the cost vector c are stored as binary vectors in these files.

There is one final issue to discuss in connection with nonduplicates. Since no duplicates were initially found across section boundaries, duplicate columns were removed from $\mathscr{W}$ before the problem (5) was solved. Hence, the effective number of columns added is unpredictable and decreases as s increases. As a consequence, the following instrumentation was added. On the first iteration of the sifting procedure, t^1/s columns are chosen from each section. Duplicate columns are removed from $\mathscr{P}$ and the number of new nonduplicate columns added, p^1, is computed. The value $r = p^1/t^1$ is then determined. Successive target sizes t^k for $k > 1$ are then adjusted to new values $\bar{t}^k = t^k/r$ to regulate the growth in $\mathscr{W}$. On each further

Table III
Problem Statistics

Problem Name	Original Columns	Nonduplicate Columns	Number of Sections s	Optimal Objective
1mil	1,000,000	673,310	4	54325.363
2mil	1,999,998	1,341,904	8	51725.598
4mil	3,999,998	2,682,175	16	50413.578
8mil	7,999,998	5,374,005	32	48994.020
13mil	12,753,313	8,549,879	51	48400.129

iteration, $\bar{t}^k/s$ columns are then chosen from each section using the procedure described in Section 2.2.

3. USING CPLEX

To test the application of the simplex method in the sifting procedure, the **CPLEX** optimizer (Bixby), version 2.0, was used. Much effort has been put into **CPLEX** to get good vectorization performance from various aspects of the code. Some modifications of **CPLEX** were made for this application. To conserve approximately 6% of CPU time, routines accessing the matrix $A_{\mathscr{W}}$ assumed that nonzeros in the matrix had the value 1. In addition, because the bases of the problem have extremely dense factorizations, the refactorization interval was set at 175.

Extensive experimentation with **CPLEX** on various versions of these problems reveal that steepest-edge pricing is more appropriate than standard reduced-cost pricing for both the primal and dual simplex methods. At the beginning of the sifting procedure, the initial feasible solution consisting of the first 1,000 columns is highly degenerate and no starting basis is available. However, the basis consisting of the m artificial variables is dual feasible, making the dual simplex method effective for solving the first few subproblems. Since a large number of columns are added on these initial iterations, using an advanced starting basis with the dual simplex method is not appropriate due to the large dual infeasibility incurred by the addition of the new columns. After the initial degenerate feasible point is passed by the sifting procedure, the optimal bases found by solving (5) are less degenerate, and a primal simplex method using the previously optimal basis works well. Note that this procedure for switching from the dual simplex method to the primal simplex method is unstable on smaller instances of the problem because the initial degenerate point may not be passed. In this case, (5) is re-optimized from an artificial start each time, and the pricing information varies due to multiple dual optimality, as discussed in Section 2.2.

The use of an artificial variable basis to start the dual simplex algorithm provides some insight about the dual variables π when using the simplex algorithm to solve the subproblems. For these problems, it was found that the optimal bases tended to have a large number of artificial variables, which are degenerate. (For the largest problem (13mil), 96 artificial variables are basic at the optimal solution.) Artificial variables also affect the prices for the sifting procedure. If the artificial variable on row i is basic, it is easy to see that $\pi_i^* = 0$, and hence, row i contributes nothing to the selection of new columns. Thus, it is worth considering the assignment of a penalty M as the cost of each artificial variable, as in the **OB1** runs (Section 4). In this case, the artificial variable may still remain basic with $\pi_i = M$. For the simplex method, no difference was found in the performance of the sifting procedure when a penalty $M = 10{,}000$ was used as compared to no penalty.

With **CPLEX**, the deletion strategy aimed at keeping the subproblems relatively small. A large number of columns requires more simplex iterations as well as more work per iteration because full pricing is being used. The size of a problem was limited to 33,000 columns. If a problem exceeded that size, then all columns $j \in \mathscr{W}$ with a reduced cost $\bar{c}_j > 100$ were deleted. (For example, in Table V on the third major iteration, 18,666 columns were added to a problem of size 18,977. Then 17,843 columns were removed because the problem was too large. This was followed by the removal of 8,884 duplicate columns.) Furthermore, as soon as an optimal solution to (5) was found with a smaller objective value than the initial feasible solution, columns were purged (using the same rule). Since the sifting algorithm used the primal simplex algorithm with steepest-edge pricing after this point, the philosophy is to keep the subproblems small.

3.1. CPLEX: Computational Results

The sequence $(t^1, t^2, \ldots, t^{10})$ was taken to be (16,000, 16,000, 8,000, 8,000, 4,000, 2,000, 2,000, 1,000, 1,000, 500), with $t^k = 1{,}000$ for $k > 10$. After removal of duplicates, the first linear program solved in each case had 851 columns. Table IV contains the summary of the results with **CPLEX**. The columns indicating the size of $\mathscr{W}$ represent the maximum and final sizes of the linear programs (5) that were solved. The number of major iterations is the number of iterations in the sifting procedure. The number of **CPLEX** iterations is the total number of simplex iterations needed to solve each subproblem. CPU time represents the CRAY Y-MP time to solve the problem, broken into miscellaneous time (for example, the time to remove duplicates from each subproblem), pricing time and optimization time. The pricing time includes the time required to add the columns in $\mathscr{P}$ to the **CPLEX** data structures.

Table V contains an iteration log for solving the problem 13mil. The second column indicates the size of $\mathscr{W}$ on each iteration. The **CPLEX** algorithm is the dual simplex or primal simplex method, each with steepest-edge pricing. The number of **CPLEX**

Table IV
Results With **CPLEX**

Problem Name	Size of $\mathscr{W}$		Major Iterations	CPLEX Iterations	CPU Time (Secs.)			
	Max.	Final			Misc.	Price	Optimize	Total
1mil	24,637	7,778	9	11,411	1.3	3.0	176.0	180.3
2mil	24,911	7,579	9	12,617	1.4	5.8	199.7	206.9
4mil	25,275	14,256	11	16,769	2.3	13.5	293.0	308.8
8mil	26,539	6,826	12	20,297	1.9	31.3	325.6	358.8
13mil	26,637	7,556	12	19,396	2.2	47.4	333.9	383.5

Table V
Iteration Log for **CPLEX** on 13mil

Major Iteration	Columns in Problem	CPLEX Algorithm	CPLEX Iterations	Objective Value	Optimize Time	Price Time	Columns Added	Columns Purged	Duplicates Deleted
1	851	Dual	108	57448.000	0.190	5.960	15,963	0	9,122
2	7,692	Dual	346	57448.000	1.500	4.581	37,332	7,080	18,967
3	18,977	Dual	1,654	57448.000	21.497	4.625	18,666	17,843	8,884
4	10,916	Dual	1,544	57448.000	16.993	4.795	18,666	0	8,595
5	20,987	Dual	3,981	57448.000	69.532	3.888	9,333	0	3,683
6	26,637	Dual	6,218	52035.965	129.514	3.584	4,641	24,725	2,267
7	4,286	Primal	1,829	49835.305	26.073	3.529	4,641	0	2,272
8	6,655	Primal	1,876	48856.961	35.269	3.500	2,049	0	1,309
9	7,395	Primal	1,356	48438.438	23.684	3.466	300	0	158
10	7,537	Primal	432	48400.691	7.333	3.288	34	0	17
11	7,554	Primal	50	48400.129	1.455	3.100	2	0	0
12	7,556	Primal	2	48400.129	0.839	3.076	0	0	0

iterations is the total number of iterations to solve the corresponding subproblems. The objective value is the final objective value of the subproblem (5). The optimization time is the time required by **CPLEX** to solve each subproblem, while the pricing time is the time required to price all 12.75 million columns and add columns to the **CPLEX** data structure.

4. USING OB1

The implementation of **OB1** (Lustig, Marsten and Shanno 1990a) of the primal-dual predictor-corrector interior point method used for these experiments was modified to take advantage of the special characteristics of the problem. In each iteration of the interior point method, it is necessary to compute $\pi^T A_{\mathscr{W}}$ twice. To do this efficiently, each subproblem was preprocessed to order the columns by column count, using the same method as described in Section 2.1. In addition, all references in the code to specific nonzero values in $A_{\mathscr{W}}$ were eliminated because the problem is assumed to have all nonzero values in $A_{\mathscr{W}}$ equal to 1.

All interior point methods for linear programming form a matrix of the form $(A\Theta A^T)$ on each iteration, where Θ is some diagonal matrix. This step is followed by a Cholesky factorization. For better vectorization performance, it was found that reordering $A_{\mathscr{W}}$ to increase the sparsity of the Cholesky factor L was not worth the effort as $(A_{\mathscr{W}}\Theta A_{\mathscr{W}}^T)$ is almost totally dense. Hence, reordering was not used, and a special version of a Cholesky factorization that assumed a dense matrix replaced the default sparse factorization routines. This factorization routine achieved an average performance on the CRAY Y-MP of 240 Megaflops (millions of floating point operations per second) on a single processor capable of a theoretical maximum of 333 Megaflops.

An analysis of the performance of **OB1** solving the subproblems revealed that the formation of $(A_{\mathscr{W}}\Theta A_{\mathscr{W}}^T)$ was requiring a significant amount of the processor time. An analysis of this computation led to an algorithm that vectorizes well for this task. For this discussion, let $\hat{n}$ represent the number of variables in the linear program (5) and let $\hat{A} = A_{\mathscr{W}}$. Then

$$\hat{A}\Theta\hat{A}^T = \sum_{j=1}^{\hat{n}} \Theta_j \hat{A}_j \hat{A}_j^T. \tag{11}$$

Note that the nonzero components of $(\hat{A}_j\hat{A}_j^T)$ are exactly equal to one. The matrix $(\hat{A}\Theta\hat{A}^T)$ is symmetric so that only the diagonal and the lower triangular part needs to be computed. Suppose that $\hat{A}_j$ has l_j nonzeros

at positions $i_{1j}, i_{2j}, \ldots, i_{lj}$. The diagonal $D \in \Re^m$ of $(\hat{A}\Theta\hat{A}^T)$ is computed as follows:

Procedure compute_diagonal

$D \leftarrow 0$.
for $j = 1$ to $\hat{n}$ **do**
  for $k = 1$ *to* l_j **do**
    $D_{i_{kj}} \leftarrow D_{i_{kj}} + \Theta_j$.

The inner loop of this procedure vectorizes (although with short vector lengths, adversely affecting peak performance) since the values $i_j, i_{2j}, \ldots, i_{lj}$ are unique.

To efficiently compute the lower triangular part of $(\hat{A}\Theta\hat{A}^T)$, an address list similar to the one described by Adler et al. (1989) is used. However, the nature of the matrix dictates that the "elementary products" are easier to store because each is equal to 1.0. To achieve the best possible vectorization, the address list is computed in a special way. Let $Q \in \Re^{m\times m}$ be a two-dimensional lower triangular matrix with a zero diagonal. For $i_1 > i_2$, define $Q_{i_1 i_2}$ to be the number of columns in $\hat{A}$ containing both i_1 and i_2. It is easy to see that the lower triangle of Q is equal to the lower triangle in the matrix $(\hat{A}\hat{A}^T)$. Let

$$\hat{Q} = \max\{Q_{i_1 i_2}: 1 \leq i_2 < i_1 \leq m\}, \tag{12}$$

so that $\hat{Q}$ is the maximum element in Q. Define T_q, for $1 \leq q \leq \hat{Q}$, by

$$T_q = \sum_{1\leq i_2<i_1\leq m} I(Q_{i_1 i_2} \geq q), \tag{13}$$

where I is an indicator function defined by

$$I(Q_{i_1 i_2} \geq q) = \begin{cases} 1 & \text{if } Q_{i_1 i_2} \geq q; \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Hence, T_q is a count of the number of elements in Q greater than or equal to q.

The address list R consists of $\hat{Q}$ sections, each with T_q ordered triples, $1 \leq q \leq \hat{Q}$. The ordered triples (i_r, k_r, j_r) in section R_q, $1 \leq r \leq T_q$, are defined by an inductive rule such that $(i_r, k_r, j_r) \in R_q$ if and only if the following three conditions are satisfied:

1. $Q_{i_r k_r} \geq q$;
2. $\hat{A}_{i_r j_r} = 1$ and $\hat{A}_{k_r j_r} = 1$, and;
3. $(i_r, k_r, j_r) \notin R_t$ for $t < q$.

Hence, R_1 is a list of all of the nonzero positions in Q, with each element of Q having one of the columns j_r that contribute to it. The list R_2 is a list of all of the nonzero positions that have two or more columns contributing to Q, using a different column from the one present in R_1.

Given the address lists, it is easy to then compute the lower triangular part of $(\hat{A}\Theta\hat{A}^T)$ using the following procedure:

Procedure compute_AAT

$(\hat{A}\Theta\hat{A}^T)_{i_1 i_2} \leftarrow 0$ for $1 \leq i_2 < i_1 \leq m$.
for $q = 1$ to $\hat{Q}$ **do**
  for $r = 1$ to T_q **do**
    $(\hat{A}\Theta\hat{A}^T)_{i_r k_r} \leftarrow (\hat{A}\Theta\hat{A}^T)_{i_r k_r} + \Theta_{j_r}$.

The inner loop of this procedure vectorizes well and substantially reduces the time it takes to compute $(\hat{A}\Theta\hat{A}^T)$. Note that this matrix is actually stored as a vector and that the location of element (i_r, k_r) is easily computed because $(\hat{A}\Theta\hat{A}^T)$ is assumed to be dense and is stored as a long vector. In fact, the address list stores this location. If $\mathscr{W}$ has $\bar{n}$ columns, then the length of the address list (the size of R) is expected to be $\bar{l}^2\bar{n}/2$, where $\bar{l}$ is the average of $l_1, l_2, \ldots, l_{\bar{n}}$.

When using the interior point method within sifting, it was found that deleting columns was not beneficial. Deleting many columns caused most of those columns to re-enter the problem on a later iteration. Hence, growth in the size of $\mathscr{W}$ was allowed and controlled. It seems that the dual solution π^* generated on a particular iteration k yielded important information regarding the dual feasibility of the problem. Discarding columns that forced this dual feasibility to be maintained was not useful due to the fact that the dual solution is in the relative interior of a dual face of the subproblem. This contrasts to using the simplex method, where the extreme point nature of the dual solution seemed to allow column deletion without any difficulty.

The predictor-corrector interior point method was started with a primal solution of $x_j = 100.0$ for each $j \in \mathscr{W}$, with a dual feasible solution of $\pi = 0$ and $z = c_{\mathscr{W}}$. The large value of x_j improves the performance of the predictor-corrector method. No attempts at implementing a restart strategy along the lines of Lustig, Marsten and Shanno (1990b) or Mitchell (1990) were successful. This is viewed as an avenue for future research.

Since these problems are highly rank deficient, a set of m artificial columns with a cost of 10,000 were included in the problems solved by **OB1** on each iteration of the sifting procedure. While the presence of these columns assists in keeping $(\hat{A}\Theta\hat{A}^T)$ better conditioned, these columns also influence the pricing information obtained by **OB1**. As remarked earlier, the presence of these columns did not affect the performance of the sifting procedure with **CPLEX**.

Table VI
Results With **OB1**

Problem Name	Final Size of $\mathscr{W}$	Major Iterations	OB1 Iterations	CPU Time (Secs.)			
				Misc.	Price	Optimize	Total
1mil	28,211	8	197	1.5	3.6	230.2	235.3
2mil	32,541	8	210	1.7	7.0	257.8	266.5
4mil	36,415	8	204	1.9	14.4	251.6	267.9
8mil	41,680	9	242	2.4	29.5	306.6	338.5
13mil	52,368	9	239	3.0	48.8	326.7	378.5

4.1. OB1: Computational Results

In this section, computational results are presented for the sifting procedure using **OB1** to solve the subproblems. The values used were $t^1 = 17{,}000$, $t^2 = 5{,}800$ and $t^k = 11{,}000$ for $k \geq 3$. Table VI presents the results for the sifting procedure using the interior point method to solve the subproblems. The final size of $\mathscr{W}$ is also the maximum size since no deletions were done on any of the problems. The number of major iterations remained essentially constant. The times represent similar values as in Section 3.1.

Table VII illustrates the iteration log for solving the problem 13mil using **OB1** as the solver. It is interesting to note that the sifting procedure with **OB1** requires fewer iterations to get past the objective value corresponding to the initial point. The prices obtained by **OB1** are insensitive to the degeneracy of the initial problem, although they are sensitive to the initial interior point used by the primal-dual algorithm. The time to solve each problem rises with the number of columns and is fairly predictable. The pricing time per iteration for this method is higher due to a higher overhead required to add columns to the **OB1** data structures than the **CPLEX** data structures.

5. A HYBRID METHOD

A careful examination of the above computational results reveals that the interior point method spends a predictable amount of time solving each subproblem. It performs quite well in the beginning, but toward the end, the same amount of time is spent to solve a new subproblem when only a few columns are added. In contrast, the simplex method has difficulty making progress away from the initial highly degenerate solution, but as fewer columns are added after this point is passed, the simplex method spends little time reoptimizing the new problem starting from an optimal basis for the previous subproblem. This led to the development of a hybrid procedure, where the interior point method solves five linear programs and is then followed by the application of the simplex method. In effect, **OB1** is being used to find a good starting point for the simplex method.

A difficulty with this method is the identification of a starting basis for the simplex method from an interior point solution. If all columns with $x_j > \epsilon$, where ϵ is a suitably chosen tolerance, are selected to be basic columns, and the dual solution is ignored, too much information is lost. Essentially, the solution provided

Table VII
Iteration Log for **OB1** Solving 13mil

Major Iteration	Columns in Problem	OB1 Iterations	Objective Value	Optimize Time	Price Time	Columns Added	Duplicates Deleted
1	1,688	9	57448.000	8.528	7.191	16,983	10,934
2	7,737	13	57448.000	13.317	6.104	16,269	7,453
3	16,553	18	57448.000	19.467	6.378	30,855	13,898
4	33,510	32	52316.105	41.113	5.745	30,855	12,791
5	51,574	34	48815.031	48.720	5.662	1,747	1,044
6	52,277	33	48431.762	48.084	5.620	244	169
7	52,352	33	48400.703	50.498	4.160	26	15
8	52,363	32	48400.129	46.224	4.752	37	32
9	52,368	35	48400.129	50.736	3.196	0	0

by the interior point method indicates something about the dual feasibility of all the columns in $\mathscr{W}$, and discarding columns for which dual feasibility is attained does not work well.

Therefore, the procedure described by Megiddo (1991) was used to identify an optimal basis. The implementation of this procedure is described by Lustig (1992). First, the procedure identifies a crash basis, leaving all nonbasic variables at the value determined by the interior point method. This solution is purified to a primal optimal solution by lowering each positive nonbasic x_j to its bound. The work per iteration of this method is equivalent to the work per iteration of the primal simplex method. The second part of the procedure maintains the dual feasibility of the interior point method by doing a similar procedure to the dual linear program. Each of these iterations is equivalent to a step of the dual simplex method. The procedure is guaranteed to converge in at most m pivots. For the results reported here, the routines implemented by Lustig, using XMP (Marsten 1981) and LA05 (Reid 1982), were employed. Neither of these codes are suitable for good vectorization. Better results would be obtained by implementing this crossover algorithm within CPLEX.

Purifying the primal and dual optimal solutions in this way maintained dual feasibility relative to the columns already in $\mathscr{W}$. All columns with $\bar{c}_j \leq 100$, $j \in \mathscr{W}$, were then passed to the simplex method, effectively paring the problem down to an acceptable size. This procedure worked well and substantially reduced the total computation time.

5.1. OB1 and CPLEX: Computational Results

In this section, computational results are presented for the hybrid scheme described above. For each problem solved, **OB1** was used to solve the first five subproblems, with $t^1 = 25{,}500$ and $t^k = 11{,}000$ for $2 \leq k \leq 4$. (Note that these target values differ from those used in Section 4.1, where total problem growth is a more serious consideration.) After the fifth linear program was solved, the crossover scheme was invoked followed by one solve of a small linear program via **CPLEX** to get an initial factorization. After this initial solve, $t^6 = t^7 = 800$ and $t^k = 400$ for $k \geq 8$. Note that t^5 is considered irrelevant due to the crossover scheme. Table VIII shows various statistics for this approach, while Table IX shows CPU time. In Table VIII, the size of the last linear program solved by **OB1** is given, along with sizes for the first and last linear programs solved by **CPLEX**. The total number of major iterations includes five interior point iterations followed by a sequence of **CPLEX** solves. The crossover iteration is not counted as a major iteration. In Table IX, the miscellaneous time includes miscellaneous tasks such as the deletion of duplicates before each subproblem is solved. The pricing time is the time to price out all the columns including the time required to add the columns to the appropriate data structures for **OB1** or **CPLEX**. The crossover time is the time used by the **OB1**/XMP routines to compute an optimal basis. The final **CPLEX** time is the time used by **CPLEX** to finish the procedure.

Table X gives an iteration log for solving the problem 13mil with the hybrid scheme. After **OB1** solves the first five subproblems, the crossover procedure is invoked. The crossover requires only 327 iterations, but as previously noted, these iterations are expensive because of the lack of vectorization in XMP and LA05. **CPLEX** is then handed a small problem and an optimal basis. This basis is factorized and checked for optimality. **CPLEX** then solves the remainder of the subproblems.

The computational performance of the hybrid scheme is clearly superior to the pure interior point or pure simplex procedures and could be improved further by doing the crossover procedure within **CPLEX**. Moreover, the hybrid procedure is clean and a more robust product. It is not subject to the unpredictable behavior of the simplex method in the earlier

Table VIII
Statistics for Hybrid Scheme

Problem Name	Size of $\mathscr{W}$			Iterations			
	Last **OB1**	First **CPLEX**	Last **CPLEX**	Major	**OB1**	Crossover	**CPLEX**
1mil	33,456	2,457	3,899	9	119	335	228
2mil	37,185	2,984	4,250	9	116	421	358
4mil	41,479	3,539	4,592	9	116	387	478
8mil	42,176	4,228	6,774	12	126	336	1,389
13mil	45,083	5,217	6,980	10	111	327	1,268

Table IX
CPU Times for Hybrid Scheme

Problem Name	CPU Time (Secs.)					
	Misc.	Price	OB1	Cross	CPLEX	Total
1mil	1.3	3.1	145.4	21.1	8.7	179.6
2mil	1.4	6.0	145.4	22.9	10.7	186.4
4mil	1.6	12.3	146.8	24.7	13.8	199.2
8mil	1.9	31.9	163.1	25.9	35.9	258.7
13mil	2.0	42.6	140.3	30.5	33.6	249.0

Table X
Iteration Log for Problem 13mil With Hybrid Scheme

Major Iteration	Columns in Problem	Method	Iterations	Objective Value	Optimize Time	Price Time	Columns Added	Duplicates Deleted
1	1,688	**OB1**	9	57448.000	8.549	7.262	25,449	14,638
2	12,499	**OB1**	14	57448.000	15.324	6.041	25,857	12,135
3	26,221	**OB1**	24	57448.000	28.582	6.493	25,857	12,774
4	39,304	**OB1**	33	50734.910	44.399	5.750	13,018	7,239
5	45,083	**OB1**	31	48530.461	43.398	0.000	0	0
*	45,083	Crossover	327	48530.461	30.538			
6	5,217	**CPLEX**	0	48530.461	3.105	3.555	1,832	1,000
7	6,049	**CPLEX**	297	48510.117	9.572	3.532	1,798	1,022
8	6,825	**CPLEX**	648	48418.199	13.754	3.571	382	238
9	6,969	**CPLEX**	316	48400.129	6.244	3.259	18	7
10	6,980	**CPLEX**	7	48400.129	0.972	3.145	0	0

"degenerate" phase, and is not sensitive to the total number of iterations at the end in the same way as **OB1**, where an additional pass can cost almost one minute of additional computation time.

6. CONCLUDING REMARKS

Recent progress in the implementations of simplex and interior point methods as well as advances in computer hardware have extended the size of linear programs that can be solved efficiently with today's computing technology. In this paper, we have shown how the best features of each of these methods can be combined to produce a very effective algorithm for solving a truly large-scale linear programming problem. It is our hope that this work motivates further research along the same lines. Of course, in the context of the present application, our work is but a first step. The ultimate goal is to solve the corresponding large-scale integer program.

ACKNOWLEDGMENT

The research of the first author was partially supported by National Science Foundation grant CCR-8815914 and Air Force Office of Scientific Research grant AFOSR-90-0273. Irvin Lustig was a Visiting Scholar at the Center for Research in Parallel Computation at Rice University. The research of Roy Marsten was partially supported by Office of Naval Research grant N09014-88-16-0104. The work of David Shanno was partially supported by the AFOSR, Air Force System Command grant AFOSR-87-0215.

REFERENCES

ADLER, I., N. KARMARKAR, M. G. RESENDE AND G. VEIGA. 1989. Data Structures and Programming Techniques for the Implementation of Karmarkar's Algorithm. *ORSA J. Comput.* **1**, 84–106.

BARUTT, J., AND T. HULL. 1990. Airline Crew Scheduling: Supercomputers and Algorithms. *SIAM News* **23**, 1, 20–22.

BIXBY, R. E. 1990. Implementing the Simplex Method: The Initial Basis. Technical Report TR90-32, Department of Mathematical Sciences, Rice University, Houston, Texas.

FORREST, J. J. 1989. Mathematical Programming With a Library of Optimization Subroutines. Presented at the ORSA/TIMS Joint National Meeting, New York (October).

FORREST, J. J., AND J. A. TOMLIN. 1990. Vector Processing in Simplex and Interior Methods for Linear Programming. *Ann. Opns. Res.* **22,** 71–100.

GERSHKOFF, I. 1989. Optimizing Flight Crew Schedules. *Interfaces* **19,** 29–43.

GOLDFARB, D. 1976. Using the Steepest-Edge Simplex Algorithm to Solve Sparse Linear Programs. In *Sparse Matrix Computations,* J. Bunch and D. Rose (eds.). Academic Press, New York, 227–240.

GOLDFARB, D., AND J. K. REID. 1977. A Practicable Steepest-Edge Simplex Algorithm. *Math. Prog.* **12,** 361–371.

LUSTIG, I. J. 1992. An Implementation of a Strongly Polynomial Time Algorithm for Basis Recovery. Manuscript, Department of Civil Engineering and Operations Research, Princeton, N. J. (in preparation).

LUSTIG, I. J., R. E. MARSTEN AND D. F. SHANNO. 1990a. On Implementing Mehrotra's Predictor-Corrector Interior Point Method for Linear Programming. *SIAM J. Optim.* **2**(3), 435–449.

LUSTIG, I. J., R. E. MARSTEN AND D. F. SHANNO. 1990b. Starting and Restarting the Primal-Dual Interior Point Method. Technical Report SOR 90-14, Department of Civil Engineering and Operations Research, Princeton University, Princeton, N. J.

MARSTEN, R. E. 1981. The Design of the XMP Linear Programming Library. *ACM Trans. Math. Software* **7,** 481–497.

MEGIDDO, N. 1991. On Finding Primal- and Dual-Optimal Bases. *ORSA J. Comput.* **3,** 63–65.

MITCHELL, J. E. 1990. An Interior Point Column Generation Method for Linear Programming With Shifted Barriers. RPI Technical Report 191, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, N.Y.

NEMHAUSER, G. L., AND L. A. WOLSEY. 1988. *Integer and Combinatorial Optimization.* John Wiley, New York.

REID, J. K. 1982. A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases. *Math. Prog.* **24,** 55–69.

ZENIOS, S. A., AND J. M. MULVEY. 1986. Nonlinear Network Programming on Vector Supercomputers: A Study of the CRAY X-MP. *Opns. Res.* **34,** 667–682.