Computational Experience with a Preconditioner for Interior Point Methods for Linear Programming

A.R.L. Oliveira and D.C. Sorensen

CRPC-TR97772
November 1997

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted September 1998; Available as Rice CAAM TR97-28; In review for *SIAM J. Optimization*

COMPUTATIONAL EXPERIENCE WITH A PRECONDITIONER FOR INTERIOR POINT METHODS FOR LINEAR PROGRAMMING

A. R. L. OLIVEIRA* AND D. C. SORENSEN†

Abstract. In this paper, we discuss efficient implementation of a new class of preconditioners for linear systems arising from interior point methods. These new preconditioners give superior performance near the solution of a linear programming problem where the linear systems are typically highly ill-conditioned. They rely upon the computation of an LU factorization of a subset of columns of the matrix of constraints. The implementation of these new techniques require some sophistication since the subset of selected columns is not known a priori. The conjugate gradient method using this new preconditioner compares favorably with the Cholesky factorization approach. The new approach is clearly superior for large scale problems where the Cholesky factorization produces intractable fill-in. Numerical experiments on several representative classes of linear programming problems are presented to demonstrate the promise of the new preconditioner.

Key words. linear programming, interior-point methods, preconditioners, augmented system

AMS subject classifications. 65F10, 65F50, 90C05, 90C06

Abbreviated Title: Computational Experience with a Preconditioner.

1. Introduction. In [9] a new class of preconditioners for for the linear systems from interior point methods for linear programming is proposed and its theoretical properties are discussed. This class avoids computing the Schur complement matrix. Instead, these preconditioners rely upon an LU factorization of a subset of columns of the constraint matrix. In this work we present several techniques for an efficient implementation of these preconditioners. Among these, are techniques for the efficient utililization of the nonzero structure of the matrix to speed up the numerical factorization. We investigate the performance of some important large scale problem cases using this preconditioner with the conjugate gradient method. Performance of this iterative approach is compared with the performance of the direct Cholesky factorization technique.

We use the following notation throughout this work. Lower case Greek letters denote scalars, lower case Latin letters denote vectors and upper case Latin letters denote matrices. Components of matrices and vectors are represented by the corresponding Greek letter with subscripts. The symbol 0 will denote the scalar zero, the zero column vector and the zero matrix, its dimension will be clear from context. The identity matrix will be denoted by I, a subscript will determine its dimension when it is not clear from context. The Euclidean norm is represented by $\|\cdot\|$ which will also represent the 2-norm for matrices. The relation $X = diag(x)$ means that X is a diagonal matrix whose the diagonal entries are the components of x. On the other hand, $diag(A)$ means the column vector formed by the diagonal entries of A. A superscript k for a scalar, vector or matrix will denote their value at the kth step of an iterative procedure.

* State University of Campinas UNICAMP, São Paulo 13083-970 (aurelio@densis.fee.unicamp.br)
This work was supported in part by the Brazilian Council for the Development of Science and Technology CNPq, and by FAPESP Fundação de Amparo à Pesquisa do Estado de São Paulo,

†Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 77005-1892 (sorensen@caam.rice.edu). This work was supported in part by NSF cooperative agreement CCR-9120008, and by ARPA contract number DAAL03-91-C-0047 (administered by the U.S. Army Research Office) .

2. Linear Programming Problems. Consider the linear programming problem in the *standard form*:

$$\begin{array}{ll} \text{minimize} & c^t x \\ \text{subject to} & Ax = b, \quad x \geq 0, \end{array} \tag{2.1}$$

where A is a full row rank $m \times n$ matrix and c, b and x are column vectors of appropriate dimension. Associated with problem (2.1) is the dual linear programming problem

$$\begin{array}{ll} \text{maximize} & b^t y \\ \text{subject to} & A^t y + z = c, \quad z \geq 0, \end{array} \tag{2.2}$$

where y is a m-vector of free variables and z is the n-vector of dual slack variables.

The optimality conditions for (2.1) and (2.2) can be written as a nonlinear system of equations with some nonnegative variables:

$$\begin{pmatrix} Ax - b \\ A^t y + z - c \\ XZe \end{pmatrix} = 0, \quad (x, z) \geq 0, \tag{2.3}$$

where $X = diag(x)$ and $Z = diag(z)$ and e is the vector of all ones.

The majority of the primal-dual interior point methods found in the literature can be seen as variants of Newton's method applied to the optimality conditions (2.3). Bellow we give a slightly more general framework:

METHOD 2.1 (Primal-Dual Method). *Given* y^0 *and* $(x^0, z^0) > 0$.
For $k = 0, 1, 2, \ldots,$ *do*
(1) *Set* $\gamma^k = (x^k)^t z^k$ *and choose* $\tau^k \in (0, 1)$ *and* μ^k.
(2) *Compute the search directions* $\Delta x^k, \Delta y^k$ *and* Δz^k *using* μ^k.
(3) *Choose a step length* $\alpha^k = \min(1, \tau^k \rho_p^k, \tau^k \rho_d^k)$ *for* $\tau^k \in (0, 1)$ *where*

$$\rho_p^k = \frac{-1}{\min_i \left(\frac{\Delta x_i^k}{x_i^k} \right)} \quad \text{and} \quad \rho_d^k = \frac{-1}{\min_i \left(\frac{\Delta z_i^k}{z_i^k} \right)}.$$

(4) *Form the new iterate*

$$(x^{k+1}, y^{k+1}, z^{k+1}) = (x^k, y^k, z^k) + \alpha^k (\Delta x^k, \Delta y^k, \Delta z^k).$$

The way to compute the search directions determine different methods. In practice it is better to separate the step length for the primal and dual problems. Thus, we define a primal step length $\alpha_p^k = \min(1, \tau^k \rho_p)$ and a dual step length $\alpha_d^k = \min(1, \tau^k \rho_d)$ which are used to compute

$$\begin{aligned} x^{k+1} &= x^k + \alpha_p^k \Delta x^k \quad \text{and} \\ (y^{k+1}, z^{k+1}) &= (y^k, z^k) + \alpha_d^k (\Delta y^k, \Delta z^k) \end{aligned}$$

on step (4) of the method.

2.1. Mehrotra's Predictor-Corrector Method. Soon after its appearance, the predictor-corrector variant [8] became the method of choice [7]. The predictor-corrector approach differs from the standard primal-dual method in the choice of the search directions. First it computes the affine directions

$$\begin{pmatrix} 0 & I & A^t \\ Z^k & X^k & 0 \\ A & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta \tilde{x}^k \\ \Delta \tilde{z}^k \\ \Delta \tilde{y}^k \end{pmatrix} = \begin{pmatrix} r_d^k \\ r_a^k \\ r_p^k \end{pmatrix} \tag{2.4}$$

where,

$$\begin{cases} r_d^k &= c - A^t y^k - z^k \\ r_a^k &= -X^k Z^k e \\ r_p^k &= b - Ax^k. \end{cases}$$

Then, the search directions are given by

$$\begin{pmatrix} 0 & I & A^t \\ Z^k & X^k & 0 \\ A & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta z^k \\ \Delta y^k \end{pmatrix} = \begin{pmatrix} r_d^k \\ r_m^k \\ r_p^k \end{pmatrix} \tag{2.5}$$

where,

$$r_m^k = \mu^k e - X^k Z^k e - \Delta\tilde{X}^k \Delta\tilde{Z}^k e$$

and $\mu^k = (\frac{\tilde{\gamma}^k}{\gamma^k})^2(\frac{\tilde{\gamma}^k}{n^{1.5}})$ with $\tilde{\gamma}^k = (x^k + \Delta\tilde{x}^k)^t(z^k + \Delta\tilde{z}^k)$.

2.2. The Search Directions. The key step in a given iteration in terms of computational cost is the solution of linear systems (2.4) and (2.5). Eliminating Δz^k, from system (2.5) it reduces to:

$$\begin{pmatrix} -D^k & A^t \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \end{pmatrix} = \begin{pmatrix} r_d^k - (X^k)^{-1} r_m^k \\ r_p^k \end{pmatrix} \tag{2.6}$$

where, $D^k = (X^k)^{-1} Z^k$. We refer to (2.6) as the *augmented system*. Eliminating Δx^k from (2.6) we get

$$S^k \Delta y^k = r_p^k + A((D^k)^{-1} r_d^k - (Z^k)^{-1} r_m^k). \tag{2.7}$$

In (2.7) $S^k \equiv A(D^k)^{-1}A^t$ is called the *Schur complement*. A similar reduction can be done for system (2.4).

We like to stress that D^k is a diagonal matrix with positive diagonal entries and it is the only change in the matrices of systems (2.6) and (2.7) at each iteration. Several entries of D^k converge to zero as the method approaches a solution while other entries tend to infinity.

2.3. Solving the Linear Systems. There are several approaches for computing the search directions. The most widely used is computing the Cholesky factorization of the Schur complement for solving the linear systems.

In this work, we are concerned with the solution of the augmented system by iterative methods. For this approach, it is essential to modify an ill-conditioned linear system into an equivalent better conditioned system. Otherwise, the iteration may be very slow or even fail to converge. This is done in such a way that it is easy to recover the original system solution from the modified one. The technique just described is known as preconditioning.

Consider the following situation: given $Bx = b$, we solve the equivalent linear system $M^{-1}BN^{-1}\tilde{x} = \tilde{b}$, where $\tilde{x} = Nx$ and $\tilde{b} = M^{-1}b$. The system is said to be preconditioned and $M^{-1}BN^{-1}$ is called the preconditioned matrix. Notice that usually it is not necessary to compute the preconditioned matrix since most of the iterative methods only access the matrix to compute matrix-vector products.

We say that a preconditioner is symmetric if $N^t = M$ because in that case if B is symmetric, the preconditioned matrix $M^{-1}BM^{-t}$ is also symmetric.

3. A New Class of Preconditioners. In [9] the following preconditioner for the augmented system was introduced*:

$$M^{-1}\begin{pmatrix} -D & A^t \\ A & 0 \end{pmatrix} M^{-t} = \begin{pmatrix} -I_n + D^{-\frac{1}{2}}A^tG^t + GAD^{-\frac{1}{2}} & 0 \\ 0 & -D_B \end{pmatrix} \tag{3.1}$$

where,

$$M^{-1} = \begin{pmatrix} D^{-\frac{1}{2}} & G \\ H & 0 \end{pmatrix} \tag{3.2}$$

with $G = H^t D_B^{\frac{1}{2}} B^{-1}$, $HP^t = [I\ 0]$, $AP^t = [B\ N]$ and

$$PDP^t = \begin{pmatrix} D_B & 0 \\ 0 & D_N \end{pmatrix}. \tag{3.3}$$

Since D_B is diagonal, we are concerned with the left upper block matrix

$$K = -I_n + D^{-\frac{1}{2}}A^tG^t + GAD^{-\frac{1}{2}}$$

which is indefinite. It is possible to exploit the structure of the problem even further, reducing it to a smaller positive definite system. If we expand K we obtain the following matrix

$$P^t \begin{pmatrix} I & D_B^{\frac{1}{2}} B^{-1} N D_N^{-\frac{1}{2}} \\ D_N^{-\frac{1}{2}} N^t B^{-t} D_B^{\frac{1}{2}} & -I \end{pmatrix} P. \tag{3.4}$$

Therefore, the problem can be reduced to a positive definite linear system involving either the matrix

$$I_m + D_B^{\frac{1}{2}} B^{-1} N D_N^{-1} N^t B^{-t} D_B^{\frac{1}{2}} \tag{3.5}$$

or the matrix

$$I_{n-m} + D_N^{-\frac{1}{2}} N^t B^{-t} D_B B^{-1} N D_N^{-\frac{1}{2}}. \tag{3.6}$$

In [9] it is shown that the eigenvalues of K squared are eigenvalues of the positive definite matrices.

3.1. Choosing the Set of Columns. We now discuss how to select the columns of A that form B. The type of matrices in (3.5) and (3.6) suggest a choice for the columns of B by looking at the values of D. Let $W = D_B^{\frac{1}{2}} B^{-1} N D_N^{-\frac{1}{2}}$. If we can chose the columns related to the smallest values of $diag(D)$, both WW^t and W^tW approach zero at the final iterations of the interior point method. Thus, a good strategy consists in taking the first m linearly independent columns of A ordered by the value of δ_{ii} in non decreasing order. This choice of columns tends to produce better conditioned matrices as the interior point method approaches a solution. This is because at least $n - m$ diagonal entries entries of D became large and diminish the importance from WW^t and W^tW.

Another way to obtain a good set of columns is to minimize $||W||$. This problem is hard to solve but it can be approached by a cheap heuristic, that is we choose the first m linearly independent columns of AD^{-1} with smallest 1-norm.

*From now on we drop the superscript k since we are concerned with one iteration of the primal-dual interior point method.

3.2. Equivalence to the Schur Complement. A useful fact is that the matrix (3.5) can be obtained via the Schur complement. Recall that the Schur complement is defined as $S = AD^{-1}A^t$ and that $A = [BN]P$. Thus,

$$S = BD_B^{-1}B^t + ND_N^{-1}N^t.$$

Now, pre-multiplying S by $D_B^{\frac{1}{2}}B^{-1}$ and post-multiplying by its transpose leads to

$$D_B^{\frac{1}{2}}B^{-1}SB^{-t}D_B^{\frac{1}{2}} = I_m + D_B^{\frac{1}{2}}B^{-1}ND_N^{-1}N^tB^{-t}D_B^{\frac{1}{2}}. \tag{3.7}$$

Observe that the matrix on the right hand side of (3.7) is the same as in (3.5). It can be shown that the right hand side vector for both preconditioned systems is the same. Therefore, the systems are completely equivalent. On the other hand, matrices (3.4) and (3.6) can not be obtained from the Schur complement.

From a practical point of view, we favor the use of the matrix-vector product on the left hand side of (3.7) even though it is a little more expensive, because it seems to be more stable according to preliminary experiments. Moreover, this representation of the preconditioner is well suited for use within a preconditioned conjugate gradient method.

3.3. Scaling the Columns. Looking at the expression $W = D_N^{-\frac{1}{2}}N^tB^{-t}D_B^{\frac{1}{2}}$ again, it is tempting to scale the matrix after selecting the columns of B such that $\|W\| \approx 0$. The following lemma shows that this idea is not easy to implement since the scaling will disappear from the preconditioned matrix.

LEMMA 3.1. *Consider the following scaling of the augmented system*

$$\begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} -D & A^t \\ A & 0 \end{pmatrix}\begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} C^{-1} & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Then the preconditioned matrix (3.4) is independent of the diagonal scaling matrix C.

Proof. The scaled system is given by

$$\begin{pmatrix} -CDC & \tilde{A}^t \\ \tilde{A} & 0 \end{pmatrix}\begin{pmatrix} C^{-1}\Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} Cb_1 \\ b_2 \end{pmatrix}$$

where $\tilde{A} = AC$. Its preconditioned matrix is as follows

$$P^t\begin{pmatrix} I & D_B^{\frac{1}{2}}B^{-1}ND_N^{-\frac{1}{2}} \\ D_N^{-\frac{1}{2}}N^tB^{-t}D_B^{\frac{1}{2}} & -I \end{pmatrix}P$$

where $AP^t = [BN]$. ∎

It is still possible to use the idea of scaling with good results. After computing the first LU factorization, the columns of A not selected as columns of B can be rescaled giving a new linear programming problem. With a proper choice of the scaling factor, the first linear systems will be better conditioned than before and yet the problem will not be badly scaled. Before a second LU factorization is computed, the original linear programming problem is recovered by undoing the scaling. Therefore, the linear programming problem will be properly scaled for the remaining of the procedure and the first iterations of the interior point method generate better conditioned systems which are the most difficult for this class of preconditioners.

3.4. Special Structures. If we apply the new preconditioner to problems with inequality constraints, we obtain the following preconditioned matrix

$$M^{-1}\begin{pmatrix} -D & A^t \\ A & E \end{pmatrix} M^{-t} = \begin{pmatrix} -I + D^{-\frac{1}{2}}A^tG^t + GAD^{-\frac{1}{2}} + GEG^t & 0 \\ 0 & -D_B \end{pmatrix}$$

Where E is a diagonal matrix with nonzero entries on the diagonal for each corresponding inequality constraint. The only difference to the standard form is term GEG^t on the left upper block. Expanding it gives

$$P^t\begin{pmatrix} I + D_B^{\frac{1}{2}}B^{-1}EB^{-t}D_B^{\frac{1}{2}} & D_B^{\frac{1}{2}}B^{-1}ND_N^{-\frac{1}{2}} \\ D_N^{-\frac{1}{2}}N^tB^{-t}D_B^{\frac{1}{2}} & -I \end{pmatrix} P.$$

Like before, see (3.4), the problem can be reduced to a smaller system giving the matrix

$$I_m + D_B^{\frac{1}{2}}B^{-1}(ND_N^{-1}N^t + E)B^{-t}D_B^{\frac{1}{2}}$$

which can be also related to the Schur complement. However, the $n-m$ system is no longer an option.

Besides this drawback, there are two important practical reason for adding the slack variables into the problem instead of eliminating then from the linear system. First, this preconditioner relies on the computation of the LU factorization of a linearly independent set of columns from A. By including the slack columns as part of A we may obtain very inexpensive factorizations. The best case is a problem with all inequalities where the matrix B formed by the slack columns is a permutation of the identity matrix. Second, the explicit use of the slack variables may also avoid redundant rows in A. The presence of redundant rows causes the first factorization to be very expensive. The reason is that all the columns have to be factored no matter the ordering, to detect redundant rows.

We remark that the augmented system for problems with bounded variables has the same nonzero structure of the augmented system for the standard form. Therefore, the preconditioners developed here can be applied to this type of problem without any change.

3.5. An Equivalent Preconditioner. The choices $H = [I\ \ 0]PD^{-\frac{1}{2}}$ and $G^t = (HA^t)^{-1}[I\ 0]P$ on (3.2) lead to the preconditioned system

$$M^{-1}\begin{pmatrix} -D & A^t \\ A & 0 \end{pmatrix} M^{-t} = \begin{pmatrix} -I + D^{-\frac{1}{2}}A^tG^t + GAD^{-\frac{1}{2}} & 0 \\ 0 & -I \end{pmatrix}.$$

This preconditioner is equivalent to (3.1) for all practical purposes on the context of interior point methods because

$$(HA^t)^{-1} = B^{-t}D_B^{\frac{1}{2}}$$

for this choice of H. However, on a more general context, where D is not necessarily diagonal, this preconditioner requires more computational work.

4. Computing the LU Factorization. This class of preconditioners is not competitive against the direct method approach by computing the Cholesky factorization without a careful implementation. This is due to the computation of an LU

Version	Factorizations	MFlops	Time(s)
Standard	25	393	3217
Keep LU	4	426	2162

TABLE 4.1
KEN13 New Factorization versus Keeping LU

factorization where the set of independent columns is unknown at the beginning of the factorization. This factorization may be too expensive for two reasons. First, it may generate too much fill-in. Second, it may be necessary to factor too many columns before the completion of the factorization since the dependent columns must be discarded. In this and next sections we discuss several techniques for the implementation of a competitive code.

For this application, the most economical way to compute the LU factorization is to work with one column at a time. This version of the LU factorization is sometimes called the delayed update form. It fits very well with our problem because when a linearly dependent column appears, it is eliminated from the factorization and the method proceeds with the next candidate column. No time is wasted in updating the matrix using columns which may turn out to be dependent. This procedure is repeated until m linearly independent columns are found. If A has less than m linearly independent columns, the factorization is applied to the right hand side of the linear programming problem. If it is linearly independent with respect to the set of columns, the problem is infeasible. Otherwise, the remaining rows of A are redundant and can be eliminated.

4.1. Keeping the Factorization. A nice property of this preconditioner is that we can work with the selected set of columns for some iterations. Observe that most of the work for computing the preconditioner consists of choosing the columns and computing the LU factorization which is already available.

It is important to notice that keeping the matrix B from previous iterations does not mean keeping a fixed preconditioner. The preconditioner depends on D and thus adapts to the new linear system at each iteration as D changes. Thus, this strategy provides an adaptive preconditioner at each iteration that is very inexpensive to compute. However, these preconditioners do not provide the best selection of columns after the first factorization according to the heuristic.

Table 4.1 illustrates this idea. The chosen problem is KEN13 a multi-commodity network flow problem. It can be obtained from netlib. The dimension of the linear system is 14627 after preprocessing. In column Version we have the standard approach of computing a factorization at every iteration against the idea for keeping the factorization. A new LU factorization is computed whenever the preconditioned conjugate gradient method takes more iterations than one over twenty times the dimension of the problem to achieve convergence. Notice that this version computes only 4 factorizations.

4.2. Incomplete LU Factorization. We find in practice that the LU factorization often generates too much fill-in. Usually, this is because no *a-priori* reordering procedure can be used to reduce the amount of fill-in. The columns of the matrix are not known until they are accepted. Later in this section we present techniques to alleviate this problem.

Here we discuss another possibility which is not adopted as a default option. It consists of computing an incomplete LU factorization. The standard incomplete

factorization where the nonzero structure of the original matrix coincides which the nonzero structure of the triangular matrices $L\backslash U$ does not work well for this problem. In fact, our implementation of the interior point method does not converge for any of the problems tested using this incomplete factorization. On the other hand, the use of drop tolerance seems to be a viable approach. The idea is to eliminate any entry smaller than a preset value. For a carefully chosen tolerance, this technique can be very useful and it actually gives better performance on some of the problems tested. This line of research deserves more investigation in the future.

4.3. Using Indicators. Indicators [5] can also be used to reduce work in the factorizations when the interior point method is close to a solution. An indicator is a tool for determining if a column is not a member of any optimal basis before the method converges. Such columns can thus be eliminated. In the context of this work, indicators can be used to keep these columns at the end of the list for finding an independent set of columns, saving work on the factorization. Since these columns are not being eliminated from the problem, it is possible to be less rigorous in the way they are determined without taking the risk of getting an erroneous solution to the linear programming problem. It turns out that the diagonal entries of D are valid indicators. Thus, the approach we have adopted here actually uses indicators for reordering the columns.

4.4. Avoiding Dependent Columns. A more sophisticated implementation would record a set of columns that make another column dependent. This information could then be used to save computational work in the subsequent factorizations. Any dependent column can be dropped from the current factorization whenever it appears behind the same set of independent columns on the new ordering.

In order to have an efficient search for these sets and to avoid excessive use of memory, this type of information can be stored at the bit level. Thus, if A has n columns, a set can be stored on n bits. Operating with these bits will be much faster than managing arrays of indices. Memory requirements can become a critical issue if arrays are used to store these sets for large scale problems.

4.5. Computing a Second LU Factorization. This approach provides a means to control excessive fill-in within the LU factorization and still produce a well conditioned problem to solve. The idea is to compute second factorization on the chosen set of independent columns using all of the most powerful *a-priori* techniques for computing an efficient sparse LU factorization, such as reordering the columns, finding strongly connected components, choosing the pivots with sparsity in mind, etc. Thus, the first factorization is used to select the basis and the second one is used to obtain the most efficient factorization of this selected basis.

Significant improvements have been observed with this approach. It often results in a great reduction of the floating point operations on the iterative linear system solver which compensates the extra work for computing the factorization. The effectiveness of this strategy will be clear on the numerical experiments shown later.

The following techniques are the default options for the second LU factorization on our code. We stress that it is not possible to use them on the first LU factorization because the structure of B is not known prior to the factorization.

The columns are permuted by the ordering of B^tB given by the minimum degree ordering. We included a threshold parameter for the choice of the pivot. At each step of the factorization, we chose a row permutation with the pivot being chosen among all candidates within the threshold. The one with least entries on its row for

the remaining columns of the original matrix is chosen. We are aware that there are many others powerful techniques which could be applied here. We plan to implement them in the future.

5. Identifying Symbolically Dependent Columns. In this and the next section we will present techniques which save computational work by studying the ordering of columns prior to computing the factorization.

Given an ordering of columns, we want to find the unique set of m independent columns which preserves the ordering. The brute force approach for this problem consists of computing the factorization column by column and discarding the dependent columns on the way. The strategies developed here will change the initial ordering and indicate when a column can be ignored in the factorization. It is important to note that the set of independent columns found by these techniques is the same set obtained by the brute force approach.

5.1. Previous Work. A powerful tool for reducing computation in the LU factorization is to identify columns that are *symbolically dependent*. That is, columns that are linearly dependent in structure for all numerical values of their nonzero entries. The idea is to find a set of say k columns with nonzero entries in at most $k-1$ rows. This set of columns is symbolically dependent. As we shall see, there are efficient algorithms for finding such sets.

Let us first consider square matrix for simplicity. In this situation, the problem is equivalent to permuting nonzero entries onto the diagonal. For any diagonal which a nonzero entry could not be assigned we have a symbolically dependent column. This problem is equivalent to find a matching of a bipartite graph where the rows and columns form the set of vertices and the edges are represented by the nonzero entries. This idea was first used by Duff [4] where several matching algorithms are compared. A Fortran code for the best of them is given. Our implementation is based on Duff's code.

In [3] this idea is extended to rectangular matrices. They were concerned in finding a sparse basis for the null space of the matrix. In order to obtain a sparse basis, it is necessary to to find a set of independent columns of the matrix which gives a sparse LU factorization. Thus, the columns are reordered by degree and the matching algorithm applied. As a result, it will give a set of columns, denominated here *key columns*, that are not symbolically dependent. Next, an LU factorization of the key columns is computed. If the set is found to be numerically dependent, the dependent columns are eliminated from the matrix. Then a new matching is computed and the process is repeated.

It is reported in [3] that for the problems tested, few dependent columns are found in the first factorization and that the factorization itself is cheap to compute since the columns are reordered by smallest degree. This idea cannot be applied to our problem because the LU factorization is expected to be much more expensive since the ordering of columns is independent of the degree. Thus, recomputing a factorization for another set of key columns is out of question. It is worth noting that the problems we intend to solve are much larger than the size of the reported problems.

5.2. Using Key Columns. In this work we have another use for the key columns. Our idea comes from the fact that the number of independent columns before the kth key column on the matrix is at most $k-1$. Therefore, it is possible to speed up the LU factorization whenever we compute $k-1$ independent columns

Problem		Standard	Key Columns		Matching	
Iteration	Nonzero	Updated	Updated	Skipped	Updated	Skipped
0	3474	6576	6572	4	5192	1394
6	6902	4478	3576	902	1102	3376
12	37369	3179	1819	1360	1170	2009
18	40826	3219	1614	1605	1190	2029

TABLE 5.1
TRUSS Updated and Skipped Columns

located before the kth key column in the ordering. Then, we can skip all the columns from the current one to the kth key column. This can be repeated for all the key columns of the matrix. Notice that we cannot just ignore the non-key columns from the beginning because key columns may be numerically dependent among them. This could be done in [3] because loosing a few sparse columns does not affect the amount of sparsity of the resulting matrix too much.

5.3. Matching During the Factorization. Sometimes the use of key columns does not save too much work. The reason is that often these columns are numerically dependent. Another way to save floating point operations is to compute the matching during the factorization. Thus, before we update the column with the factorization, we verify whether it is symbolically dependent or not. If it is, the column is discarded and the factorization continues with the next column.

This technique saves computational work because the matching can be done on the original matrix instead of the factored matrix. Thus, fill-in entries are not accessed. Moreover, no floating point operation is performed. If many columns are discarded, the overhead caused for the ones that are not symbolically dependent is compensated and the overall time for computing the factorization is reduced.

5.4. Preliminary Experiments with Symbolically Dependent Columns. Tables 5.1 and 5.2 show some results for the techniques presented in this section. For these experiments we used the problem TRUSS. This is a problem for minimizing the weight of structures subject to external loads. This problem is part of the netlib test collection of problems. The linear system has dimension 1000 and the problem has 8806 columns. Both tables show the number of nonzero entries for the first LU factorization for a few iterations of the interior point method. A factorization is computed at every iteration. Table 5.1 shows the number of columns that are numerically updated for the standard approach, where no symbolic technique is used. It shows the number of columns numerically updated and skipped with the key columns approach and with the matching approach that computes the matching inside the factorization. Table 5.2 has the floating point operation count for computing the factorization for the three approaches.

It is clear that the key columns and the matching approaches can lead to great savings on the total number of floating point operations on the factorization. Later we will present experiments showing that the overhead caused by these techniques is compensated by the savings on floating point operations for most test problems.

6. Symbolically Independent Columns. Just like there exist symbolically dependent columns, it is possible to define columns that are *symbolically independent* i.e., columns that are linearly independent in structure for all numerical values of their nonzero entries. This concept can be useful for computing the LU factorization of a

Iteration	Nonzero	Standard	Key Columns	Matching
0	3474	97.4	97.4	78.7
6	6902	207.9	148.9	28.5
12	37369	7143	2924	1281
18	40826	6782	2263	1239

TABLE 5.2
TRUSS Flops Count $\times 10^3$

rectangular matrix.

A strategy consists of moving the symbolically independent columns to the beginning of the ordered list since those columns are necessarily going to be on the factorization for this ordering. Then these columns can be reordered further by any known strategy such as minimum degree in order to reduce the number of fill-ins on the LU factorization. Note that only the key columns are candidate to be symbolically independent.

In [2], it is established that a set columns is symbolically independent if and only if it can be permuted into a matrix which contains an upper triangular submatrix with nonzero on the diagonal. We are not aware of any efficient algorithm for finding all the symbolically independent columns from a given ordered set. Therefore, we developed a heuristic approach to identify some of the symbolically dependent columns.

In the heuristic described below, we say that column j is the first entry column of row i if j contains the first nonzero entry in row i on the ordered set. We consider a column j symbolically independent given an ordered set if at least one of the following rules applies:

•1. Column j is the first entry column of at least one row;

•2. Column j is the second entry column of a row i and the first entry column of row i is also first entry column for at least another row not present on column j.

This set of rules guarantees that the columns selected are symbolically independent but it does not guarantee that all symbolically independent columns are found. For instance, consider the following sparse matrix

$$\begin{pmatrix} \times & \times & \times \\ \times & \times & \\ \times & & \end{pmatrix}$$

applying the heuristic, we determine that the first column is symbolically independent by rule 1 and the second by rule 2. However, we fail to notice that the third column is also symbolically independent.

6.1. The Merging Strategy. After reordering the symbolically independent columns, it may possible to reduce still further the number of fill-ins in the factorization by merging the symbolically independent and dependent list of columns using the degree as criteria. This is allowed whenever the symbolically independent columns remain so.

It is very expensive to verify whether the columns remain symbolically independent at every step of the merging process. Therefore, we use the first ordering of the columns as a cheap heuristic. Thus, we move up on the list a symbolically dependent column with lower degree provided it remains behind the symbolically independent columns with lower index on the first ordering.

6.2. Preliminary Experiments with Symbolically Independent Columns. Tables 6.1 and 6.2 show some results for the techniques presented in this section. For

Iteration	Columns	Standard	Independent	Merging
0	12761	48406	49407	49467
11	12477	56261	53564	53190
17	12352	73077	59769	59679
23	12339	63650	55278	55287

TABLE 6.1
KEN13 Number of Nonzero Entries on the Factorization

Iteration	Columns	Standard	Independent	Merging
0	12761	203	230	230
11	12477	86.2	83.3	81.9
17	12352	398	397	391
23	12339	170	130	130

TABLE 6.2
KEN13 Flops Count $\times 10^3$ *for the Factorization*

these experiments we use problem KEN13. Table 6.1 shows the number of nonzero entries for the LU factorization. Only iterations of the interior point method where the factorization is computed are shown. The number of symbolically independent columns found is shown under Columns. Three different versions of the method are compared. Here the standard approach, computes the matching during the factorization but does not find symbolically independent columns. The second technique find symbolically independent columns by the heuristic given early, move them to the front and reorder them by smallest degree. The third technique merge the symbolically independent list of columns with the other list as described before. Table 6.2 has the floating point operation count for the three versions.

It can be seen that finding and reordering symbolically independent columns can save computational work on the factorization by reducing the number of fill-in entries. The merging version also helps saving floating point operations. Notice that the first factorization (iteration 0) is very expensive in terms of floating point operations because this problem has redundant rows.

Table 6.3 present the results for problems TRUSS and KEN13 for total time in seconds and for the average number of floating point operations for solving the linear systems. Each column of the table adds one of the techniques to the previous techniques. Thus, the standard column shows the results where none of the technique using symbolic columns is used. Column key adds the key columns technique. Matching computes the matching during the factorization. Independent moves symbolic independent columns to the front and reorder them. Finally, the independent and dependent lists are merged.

It is clear from this table that saving floating point operations does not imply on saving computational time. For problems of larger dimension these techniques became more important because the number of fill-in entries tend to grow faster as dimension increases. Notice that the reordering of symbolically independent columns does not necessarily reduces the number of floating point operations. It is hard to tell beforehand if this technique will work well for given problem. In the next section we shall present numerical results for large-scale problems using the techniques developed in this work.

Problem		Technique				
Name	Results	Standard	Key	Matching	Independent	Merge
TRUSS	MFlops	86.3	78.4	76.9	85.6	85.5
	Time (s)	172	164	171	189	190
KEN13	MFlops	412	407	406	366	367
	Time (s)	2432	2487	2591	2329	2343

TABLE 6.3
Time and Flops Count for All Techniques

7. Numerical Experiments and Conclusions. In this section we present several numerical experiments with the new preconditioner. These experiments are meant to expose the type of problems where the new approach performs better. Therefore these experiments are not to be seen as a way to determine which approach for the interior point methods is better in a general context. Nevertheless, as it can be inferred by the results, the new preconditioner is an important option for some class of problems. Moreover, its importance increases when large scale problems are involved.

7.1. The Set of Test Problems. In this section we briefly describe the problems used on the numerical experiments. Problems FIT1P and FIT2P model the fitting of linear inequalities to data by minimizing a sum of piecewise-linear penalties. These problems belong to the netlib collection of linear programming problems.

The PDS model is a multi-commodity problem with 11 commodities and whose size is a function of the number of days being modeled. Thus, PDS-2 models two days, PDS-20 models twenty days and so on. A generator for this model is available and experiments for problems of a variety of sizes are presented.

DIFFICULT is the linear programming relaxation of a large network design problem. Since the edges have capacity that are integral, the original problem is an integer problem. This problem was formulated by Daniel Bienstock and supplied by Eva Lee [6].

The QAP problems are models for the linearized quadratic assignment problem [10]. A source for this type of problems is the QAPLIB [1]. The problems tested here for the QAP model are from this collection with the modification described in [10]. Since the new preconditioner obtain nice computational results for this type of problems, even for the small ones, we present results for more problems of this class.

7.2. Numerical Experiments. The following computational results compare the behavior of the direct method approach against the new preconditioner based upon the computation of an LU factorization for a subset of columns of A. The preconditioned positive definite matrix (3.7) is used for the experiments. The LU approach uses the preconditioned gradient conjugate method as solver for the linear system.

The algorithms and procedures are coded in C. The reordering function for the Cholesky factorization which was implemented by Eva Lee also in C uses the minimum degree strategy. I/O routines from the CPLEX callable library are used to input the problems in the standard MPS format.

All the experiments of this section are carried out on a Sun Ultra-Sparc station. The floating point arithmetic is IEEE standard double precision. The stopping tolerance for the interior point method and preconditioned conjugate gradient is the square root of machine epsilon. The optimality conditions are used as stopping crite-

Problem	Dimension	Number of Nonzero Entries		
		Matrix A	Schur Complement	Matrix L
FIT1P	627×1677	9868	196878	196878
FIT2P	3000×13525	50284	4501500	4501500
DIFFICULT	31427×274347	814994	706790	6089264
PDS-01	1148×3729	7440	5614	10735
PDS-02	2499×7535	15534	12036	38464
PDS-06	9068×28655	60944	46402	564505
PDS-10	15538×48670	104550	79646	1573733
PDS-15	25187×77043	165819	125354	4225522
PDS-20	33874×105728	227199	170631	6886719
PDS-25	39977×131239	281299	211509	10297297
PDS-50	79917×272091	581362	431527	41269921
CHR20A	4219×7810	27380	107509	942275
CHR20B	4219×7810	27380	107349	924810
CHR20C	4219×7810	27380	107809	1003786
CHR22A	5587×10417	36520	153856	1424837
CHR22B	5587×10417	36520	153680	1453225
CHR25A	8149×15325	53725	249324	2653126
ELS19	4350×13186	50882	137825	3763686
HIL12	1355×3114	15612	34661	611836
NUG05	89×135	495	884	2501
NUG15	2729×9675	38910	88904	2604504
ROU10	839×1765	8940	19274	242015
SCR12	1151×2784	10716	24965	334090
SCR15	2234×6210	24060	59009	1254242
SCR20	$5079X15980$	61780	166709	6350444

TABLE 7.1
Basic Statistics

ria. The parameter τ has the fixed value 0.99995. The slack variables are added into the problem explicitly.

Concerning the computation of the LU factorization for the preconditioner, the following techniques presented on previous sections are used as default:

- A new LU factorization is computed from one iteration to the other whenever
  - The preconditioned conjugate gradient method takes too many iterations to achieve convergence or;
  - The true residual for the linear system is larger than the working tolerance.
- A second LU factorization is computed whenever the first factorization is too dense;
- Key columns are used to speed up the LU factorization;
- Symbolically independent columns are moved to the front and reordered;
- The merging strategy as described before is used.

Table 7.1 contains the basic statistics about the test problems. The dimension and number of nonzero entries shown for the matrix of constraints refer to the preprocessed problems. The number of nonzero entries for the Schur complement includes only the lower half of the matrix.

Table 7.2 compares the number of iterations between the new preconditioner and the Cholesky factorization approach for the interior point method. In column Fact

Problem	Cholesky Iterations	LU Approach		
		Iterations	Fact.	Refact.
FIT1P	22	23	14	12
FIT2P	25	25	11	10
DIFFICULT	30	30	5	4
PDS-01	23	21	10	0
PDS-02	27	29	8	0
PDS-06	39	38	6	4
PDS-10	50	54	6	5
PDS-15	63	61	6	6
PDS-20	63	73	6	6
PDS-25		75	6	6
PDS-50		85	8	7
CHR20A	24	24	10	8
CHR20B	27	26	11	9
CHR20C	19	19	9	7
CHR22A	24	24	10	8
CHR22B	29	30	11	9
CHR25A	33	31	9	8
ELS19	32	31	26	24
HIL12	18	19	16	14
NUG05	7	7	8	6
NUG15	21	21	22	20
ROU10	19	19	15	13
SCR12	12	15	14	12
SCR15	23	23	22	20
SCR20	24	24	25	23

TABLE 7.2
Cholesky versus LU approach - Number of Iterations

we have the number of LU factorizations needed for the interior point methods on this strategy including a factorization for computing the starting point. In column Refact we have the number of refactorizations performed. Notice that the number of iterations for the interior point methods on both approaches is about the same for most problems. Only problem PDS-20 presented a large difference. There are no results for problems PDS-25 and PDS-50 for the Cholesky approach because it would take a large amount of time to solve this problems.

It is interesting to note that the direct approach does not obtain a clear advantage over the iterative approach for these problems. Most likely, this is due to the small tolerance chosen for the experiments. The solution obtained in these experiments agree to at least eight significant digits for all of the problems whose objective value are known to us.

Table 7.3 shows a comparison between both approaches for the total running time and number of floating point operations. The number of floating point operations shown is the average for computing and solving the linear systems for the interior point method.

For most problems tested, the LU approach takes less total time for solving the problems and less floating point operations in average. It is no surprise since these

	Cholesky		LU Approach	
Problem	Time	MFlops	Time	MFlops
FIT1P	308	87.6	19.7	4.7
FIT2P	44122	9110.3	505	106.7
DIFFICULT	51590	7335.2	29964	6115.8
PDS-01	4.63	0.3	22.3	6.8
PDS-02	23.97	1.9	102.6	22.9
PDS-06	1528	207.1	1676	237.6
PDS-10	8885	927.7	5565	529.0
PDS-15	51096	4482.6	16450	1360.0
PDS-20	103717	8911.9	33130	2275.2
PDS-25		16705.2	52676	3529.4
PDS-50		150776.2	253966	12979.7
CHR20A	1161	244.1	337	77.2
CHR20B	1189	232.6	432	93.2
CHR20C	1093	285.8	273	75.2
CHR22A	1968	414.8	618	138.6
CHR22B	2405	433.0	709	110.5
CHR25A	6501	995.4	1456	232.2
ELS19	26560	5212.0	16861	454.2
HIL12	1064	386.6	974	294.9
NUG05	0.15	0.1	0.27	0.3
NUG15	11184	3473.8	14643	3431.2
ROU10	275	98.1	259	107.2
SCR12	321	136.0	145	57.9
SCR15	3659	1008.3	1470	190.0
SCR20	47480	12331.3	25679	1397.0

TABLE 7.3
Cholesky versus LU approach - Time and Flops

models were chosen for this reason. The purpose of these experiments is to show the type of problems where the new approach is expected to perform better. For example, on problems like FIT1P and FIT2P that have dense columns, the Schur complement matrix is already very dense as it can be seen on table 7.1. Thus, this approach takes much more computational effort than the LU approach.

The QAP model problems also lead to Schur complement matrices that are not very sparse even though they are somewhat sparser than the FIT problems. This, together with the fact that the factorization generates considerable fill-in makes the Cholesky approach less effective. This becomes apparent as the size of the problems grows. The only problems for this class where the Cholesky factorization approach performs better are NUG05 which is very small and NUG15 where it was faster even though a few more floating point operations were required. The LU approach does not perform very well for this problem because a new factorization is computed for every single iteration. Notice that the opposite occur to problem $ROU10$. The new approach is faster nonetheless it takes more floating point operations.

The explanation for this apparently contradictory result seems to be the number of LU factorizations performed. This factorization have a large overhead compared with solving triangular systems. Thus, problems that need to compute few factorizations

spend less time performing work unrelated to floating point operations.

The PDS model problems do not generate dense Schur complement matrices. On the other hand, the Cholesky factorization can generate many fill-in entries. As the dimension of the problem increases, the LU factorization approach performs better compared to the direct approach. This is clearly illustrated on figures 7.1 and 7.2.

Another factor which helps the LU preconditioner obtain good results for the PDS model which is not present for the QAP model is that the number of LU factorizations is very small compared to the number of iterations. Therefore, computing a solution for the linear systems on a large number of iterations for these problems is inexpensive since no factorization is computed. This fact also applies for problem DIFFICULT explaining the good performance of the new approach even considering that the Schur complement matrix is sparse.

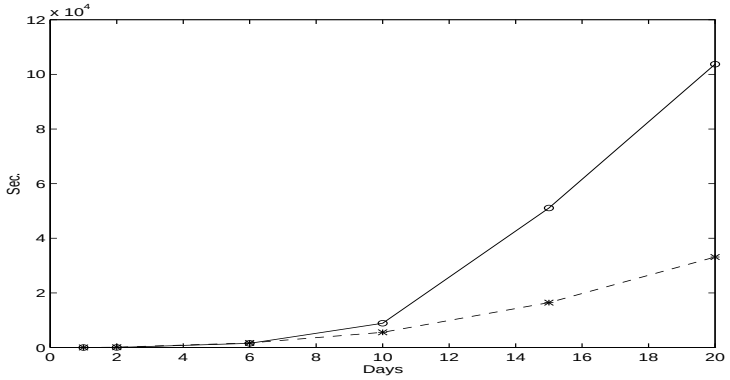

FIG. 7.1. *Running Time for PDS Model*

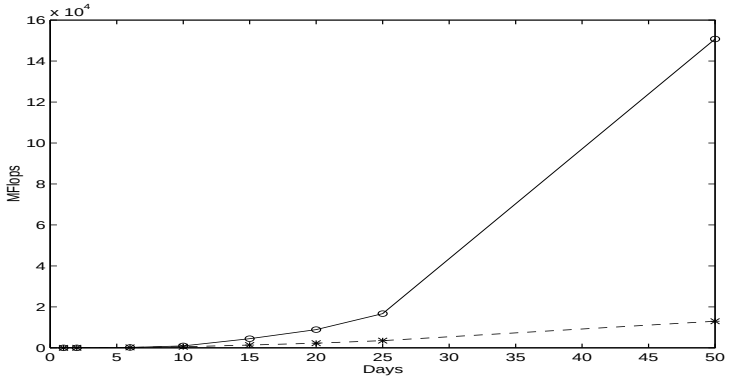

FIG. 7.2. *Flops Count for PDS Model*

Table 7.4 shows in more detail the effect of the heuristics used for keeping the LU factorization and computing a second factorization for problem DIFFICULT. Column PCG contains the number of iterations taken by the preconditioned conjugate gradient method to achieve convergence. This result is for the first linear system of the interior point method. The numbers for the second linear system are similar. The number of nonzero entries for the first and the second LU factorization are shown only for the

IP Iteration	PCG	Nonzero Entries	
		First LU	Second LU
0	901	109894	
1	897		
2	741		
3	3289		
4	267	143739	114967
5	279		
6	287		
7	336		
8	448		
9	699		
10	906		
11	1434		
12	2194		
13	277	231102	111702
14	338		
15	373		
16	404		
17	367		
18	538		
19	585		
20	753		
21	1194		
22	1235		
23	1845		
24	200	5901594	124445
25	279		
26	610		
27	211	5571610	168273
28	232		
29	221		
30	196		

TABLE 7.4
DIFFICULT Factorization Heuristics

iterations where a new factorization is computed. This number refers to the sum of nonzero entries of L and U.

Notice that only five factorizations are computed on thirty iterations of the interior point method. A second factorization is computed four times. Notice from table 7.1 that the Cholesky factorization for the Schur complement of this problem gives over six million nonzero entries.

7.3. Conclusions. This class of preconditioners behaves very well near the end of the interior point iteration. This is important because the linear systems became more ill-conditioned as an interior point iteration approaches a solution. These systems are difficult to solve by iterative methods with most previously known preconditioners. However, an efficient implementation of these preconditioners is not trivial.

We discussed how a careful implementation of this preconditioner made this approach competitive with the direct method approach. This is particularly true on classes of problems where the Cholesky factorization of the Schur complement matrix has a large number of nonzero entries.

Among the implementation details, we presented several techniques for computing the LU factorization given an ordered set of columns. Some of these techniques, such as finding symbolically dependent and independent columns, can be used in other problems of mathematical programming.

7.4. Future Work. From the point of view of improving the preconditioner, it will be interesting to develop a scaled version which may give better conditioned systems at the first iterations of the interior point method without compromising its convergence. This strategy will lead to a more robust preconditioner.

Another possibility for improvement is a more sophisticated way to decide when to keep the current LU factorization. The savings of keeping a factorization can be so high that this approach deserves more attention. The computation of the first factorization can also be improved. The approach that reorder the columns to reduce the amount of fill-in entries deserves attention. There are problems where this number can be considerably large. The ordering given by this approach is not the ideal ordering in view of the performance of the preconditioned system. Nonetheless, it is not an important drawback in comparison.

All experiments reported in this work use the square root of epsilon machine as tolerance. However, it is well known that at least on the first iterations of the interior point methods it is not necessary to find the directions with great precision. Thus, a relaxed precision will speed up solving the linear system and perhaps make the preconditioner more robust. It will be interesting to run experiments with the other preconditioned linear systems. The indefinite linear system and the $n-m$ dimensional positive definite linear system.

Finally, a more sophisticated LU factorization which includes strongly connected components and a pivot strategy for reducing the number of fill-in entries among other techniques should improve the results significantly. These techniques are to be used on the second factorization when all the columns are known.

REFERENCES

[1] R. S. Burkard, S. Karisch and F. Rendl, *QAPLIB - A Quadratic Assignment Problem Library*, European J. Oper. Res. 55 (1991), pp. 115–119.

[2] T. F. Coleman and A. Pothen, *The Null Space Problem I. Complexity*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 527–539.

[3] ———, *The Null Space Problem II. Algorithms*, SIAM J. Alg. Disc. Meth., 8 (1987), pp. 544–563.

[4] I. S. Duff, *On Algorithms for Obtaining a Maximum Transversal*, ACM Trans. Mathe. Software, 7 (1981), pp. 315–330.

[5] A. S. El-bakry, R. A. Tapia, and Y. Zhang, *A Study of Indicators for Identifying Zero Variables in Interior-Point Methods*, SIAM Rev., 36 (1994) pp. 45–72.

[6] E. K. Lee, *Private Communication.*

[7] I. J. Lustig, R. E. Marsten and D. F. Shanno, *On Implementing Mehrotra's Predictor-Corrector Interior Point Method for Linear Programming*, SIAM J. Optimization, 2 (1992), pp. 435–449.

[8] M. Mehrotra, *On the Implementation of a Primal-Dual Interior Point Method*, SIAM J. Optimization, 2 (1992), pp. 575–601.

[9] A. R. L. Oliveira and D. C. Sorensen, *A New Class of Preconditioners for Large-Scale Linear Systems from Interior Point Methods for Linear Programming*, Technical Report TR97-27, Department of Computational and Applied Mathematics, Rice University, 1997.

[10] M. Padberg and M. P. Rijal, *Location, Scheduling, Design and Integer Programming*, Kluwer Academic, Boston, 1996.