ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-227

Computational Experience with a Dense Column Feature for Interior-Point Methods

by

Marc Wenzel,* Joseph Czyzyk,† and Stephen Wright†

Mathematics and Computer Science Division

Technical Memorandum No. 227

August 1997

[illegible]

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

*On Leave from the Institut für angewandte Mathematik, Am Hubland, 97074 Würzburg, Germany [mwenzel@mathematik.uni-wuerzburg.de].

†{czyzyk,wright}@mcs.anl.gov.

Contents

Computational Experience with a Dense Column Feature for Interior-Point Methods

by

Marc Wenzel, Joseph Czyzyk, and Stephen Wright

Abstract

Most software that implements interior-point methods for linear programming formulates the linear algebra at each iteration as a system of normal equations. This approach can be extremely inefficient when the constraint matrix has dense columns, because the density of the normal equations matrix is much greater than the constraint matrix and the system is expensive to solve. In this report we describe a more efficient approach for this case, that involves handling the dense columns by using a Schur-complement method and conjugate gradient iteration. We report numerical results with the code PCx, into which our technique now has been incorporated.

1 Introduction

Primal-dual interior-point methods solve the linear programming problem

$$(LP) \qquad \min c^T x \quad \text{subject to} \quad Ax = b, x \geq 0$$

by applying Newton-like methods to the optimality conditions for this constrained problem, also known as the Karush-Kuhn-Tucker (KKT) conditions (see Wright [13]). At each interior-point iteration, a large block-structured sparse linear system with the matrix

$$\begin{pmatrix} 0 & A & 0 \\ A^T & 0 & I \\ 0 & S & X \end{pmatrix}$$

is solved in order to obtain a search direction, where S and X are diagonal matrices whose positive diagonal elements are simply the components of the vectors x and s at the current iteration. Block elimination leads to a smaller system, known as the *augmented system*, whose coefficient matrix is

$$\begin{pmatrix} 0 & A \\ A^T & -D^{-2} \end{pmatrix}, \tag{1}$$

where $D = S^{-1/2}X^{1/2}$ is also positive diagonal. A further step of block elimination yields the *normal equation* form, in which the coefficient matrix is

$$AD^2A^T. \tag{2}$$

In general, the matrix D and hence AD^2A^T vary from one iteration to the next. Factorization of this matrix is the dominant computational operation in most interior-point codes, and this operation must be performed efficiently if the code is to be effective. Since the matrix AD^2A^T is positive definite, of smaller dimension and (usually) sparse, highly developed software for sparse Cholesky factorization can be applied to the system, and ordering heuristics can be used to reduce any fill-in that may occur during the factorization process.

In some real-world problems, however, the constraint matrix A contains one or more dense columns. (Linear programs arising in stochastic programming often have this property, for example.) In these cases, the normal equations matrix AD^2A^T will generally be dense, even if the vast majority of columns in A contains just a few nonzero elements. Then is very costly to factor or even to store the matrix AD^TA^T.

Various strategies have been proposed to alleviate the problems caused by dense columns. These include

- Splitting of the dense columns;
- Sherman-Morrison-Woodbury update (SMW), also known as the Schur complement approach;
- Application of a preconditioned conjugate gradient method (pcg) to the normal equations.

In this report, we focus on the latter two methods. After reporting the state of the art, concerning dense-column features, and presenting the splitting technique in Section 2, we briefly summarize the underlying linear algebra in Section 3. The main part of the paper, Section 4, contains details of our scheme to implement a dense-column-feature combining SMW and pcg that defeats the numerical difficulties reported in Section 2. Numerical results are provided to clarify the advantages as well as the limitations of the method proposed. Finally, we point to some interesting questions that remain unresolved.

2 State of the Art

The normal equations (2) offers several advantages over the augmented system form (1):

- The normal equations have smaller dimension.
- All pivot orderings for factoring a positive definite matrix are stable, so we are free to use one of the highly developed sparse matrix ordering heuristics to reduce the amount of fill-in that occurs during the factorization.
- The ordering and allocation of data structures need to be performed just once, prior to the first iteration.

By contrast, pivot ordering in the augmented system needs to take account of issues of numerical stability, so it typically needs to be recomputed a number of times during execution of the algorithm. Fourer and Mehrotra [4] use a sparse Bunch-Parlett solver but do not explicitly account for the special block structure of the augmented system (the presence of a zero block in the upper left and a diagonal matrix in the lower right), as would be necessary to implement a method with comparable efficiency to the normal equation approach. Software for the augmented system approach is, at the time of writing, not widely available.

In [4],the authors compared their implementation of Mehrotra's algorithm with a realization of the normal equations approach with minimum degree ordering. In their experiments, the augmented system was slower by an average factor of 1.4, with no clear trend as the problem size increases. However, their normal equations code did not include special handling of problems with dense columns in the constraint matrix A, so the results were skewed by such problems as `fit1p`, `fit2p`, `israel` and `seba` in which the unmodified normal-equations approach is quite inefficient. When these problems are omitted from consideration, a few problems still remain for which the Cholesky factors are relatively dense, even though none of the columns of the matrix A are particularly dense, and the augmented system approach is superior on these examples. Still, the average ratio of CPU time for augmented systems to CPU time for the normal equation approach is around 1.6 over all problems without dense columns in A.

The splitting technique is described by Vanderbei [11]. Each dense column a_i is split into a number of columns $a_i^1, \ldots, a_i^{k_i}$ with lower density, such that $\sum_{j=1}^{k_i} a_i^j = a_i$ and $a_i^r a_i^{s^T} = 0$ for $r \neq s$. The variable x_i corresponding to a_i is split accordingly, and extra constraints are introduced to ensure that the replications of x_i have the same value at the solution. Even if the outer product $a_i a_i^T$ is completely dense, the contribution $\sum_{j=1}^{k_i} (a_i^j)^T a_i^j$ to the modified matrix is only block diagonal. To be specific, when a_i is split to k_i columns of approximately equal density, then k_i blocks of size $\frac{1}{k_i^2}\hat{m}^2$ appear. Hence, the outer product $\sum_{j=1}^{k_i} (a_i^j)^T a_i^j$ will be approximately $1/k_i$ as dense as $a_i^T a_i$.

Each splitting results in the introduction of a new variable and a new constraint via a linking matrix (see Vanderbei [11]) to the problem, resulting in a transformed problem with coefficient matrix $\hat{A} \in \mathbb{R}^{\hat{m} \times \hat{n}}$, where $\hat{m} := m + \sum(k_i - 1)$ and $\hat{n} := n + \sum(k_i - 1)$. In the most crucial *LPs* one wants to reduce the density by a factor of 100 to 1000; hence, as in the `fit`-class with > 20 dense columns, the problem size increases dramatically, making this method inefficient on large problems.

For the two alternative approaches—SMW and pcg—computational results have already been reported by other researchers. In Gill et al. [5], the authors use a pure pcg with a sparse matrix as preconditioner. They point out that an excellent preconditioner is needed to keep the number of pcg iterations at a reasonable level. This goal is important because each conjugate gradient iteration is about as expensive as a simplex step.

Adler et al. [1] rely on a similar method, but report difficulties in generating a direction precise enough for computing an accurate primal solution at termination. They use an exact factorization of the full matrix in the last iteration to accomplish this.

Choi, Monma, and Shanno [2] prefer the Sherman-Morrison-Woodbury update, yielding a direct method in place of the iterative conjugate gradient approach. They resort to iterative refinement when solving the equations during the last stages of the IPM to defeat the numerical instability incorporated in the SMW approach.

Lustig, Marsten, and Shanno [7] use Schur complements, but report problems of ill conditioning. They try to combat this by factoring $A_s D_s^2 A_s^T + \tau I$, where $\tau = \epsilon \max D_s^2$ and ϵ is a small multiple of the machine precision. Though making heavy use of iterative refinement, which was more time consuming than factoring a denser matrix, they failed to achieve more than one digit of accuracy on the `pilotja` test problem. In the final version of their code OB1, they used a default setting of `OFF` for the "no dense columns removed" option. Subsequently [8], the authors introduced a switch option that uses Schur complements as the default technique, and switches to the more expensive pcg strategy whenever the spread of the diagonal elements of the factorization is larger than 10^{14}.

3 Linear Algebra

The SMW formula for updating the inverse of a matrix $A \in \mathbb{R}^{n \times n}$ after a rank-k update UV^T is as follows:

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U[I + V^T A^{-1} U]^{-1} V^T A^{-1}.$$

This formula assumes that both A and $I + V^T A^{-1} U$ are nonsingular. Since we usually have $k \ll n$, the latter matrix usually has small dimensions. In fact, it is the Schur complement of A in the matrix

$$\begin{pmatrix} A & U \\ V^T & -I \end{pmatrix} = \begin{pmatrix} I & 0 \\ V^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -I - V^T A^{-1} U \end{pmatrix} \begin{pmatrix} I & A^{-1}U \\ 0 & I \end{pmatrix}.$$

Hence, the SMW approach is sometimes also referred to as the *Schur complement* approach.

When the approach is applied to interior-point methods, a labeling routine is used to identify the dense columns of A. Subject to some reordering of the columns (which we ignore for simplicity), the matrix is partitioned as $A = [A_s, A_d]$, where A_s contains the columns of A that are flagged as sparse and A_d consists of the dense columns. The diagonal matrix D from (1) and (2) is partitioned accordingly into D_s and D_d. The product in (2) can now be written as

$$AD^2A^T = [A_s, A_d] \begin{pmatrix} D_s^2 & 0 \\ 0 & D_d^2 \end{pmatrix} \begin{bmatrix} A_s^T \\ A_d^T \end{bmatrix} = A_s D_s^2 A_s^T + A_d D_d^2 A_d^T.$$

We use n_d to denote the number of columns in A_d and n_s as the number of columns in A_s.

We wish to factor the matrix AD^2A^T in order to solve linear equation systems with the coefficient matrix (2). We assume that a Cholesky factorization is available for the sparse part as follows:

$$P(A_s D_s^2 A_s^T)P^T = LL^T,$$

where L is lower triangular, and P is a permutation matrix usually chosen to reduce the density of L. Let $\tilde{L} = P^T L$, and let W be the solution of the system

$$\tilde{L}W = A_d.$$

(Note that W can be obtained at the cost of n_d forward substitutions with the factor L and some trivial permutation operations.) The SMW formula can now be applied to obtain the inverse of AD^2A^T as follows:

$$\begin{aligned}
&\left(A_s D_s^2 A_s^T + (A_d D_d)(A_d D_d)^T\right)^{-1} = \\
&= (\tilde{L}\tilde{L}^T)^{-1} - (\tilde{L}\tilde{L}^T)^{-1} A_d D_d [I + D_d A_d^T (\tilde{L}\tilde{L}^T)^{-1} A_d D_d]^{-1} D_d A_d^T (\tilde{L}\tilde{L}^T)^{-1} \\
&= \tilde{L}^{-T} \left\{ I - \tilde{L}^{-1} A_d (D_d^{-2} + A_d^T \tilde{L}^{-T} \tilde{L}^{-1} A_d)^{-1} A_d^T \tilde{L}^{-T} \right\} \tilde{L}^{-1} \\
&= \tilde{L}^{-T} \left\{ I - W(D_d^{-2} + W^T W)^{-1} W^T \right\} \tilde{L}^{-1}. \qquad (3)
\end{aligned}$$

The overall procedure for solving linear systems with the coefficient matrix (2) is organized as follows:

- Calculate and store $W = \tilde{L}^{-1} A_d$.
- Form the $n_d \times n_d$ matrix $D_d^{-2} + W^T W$ and compute its (dense) Cholesky factorization $L_d L_d^T$.
- Apply the formula (3) to solve the system $AD^2A^T y = r$ as follows:
  - a forward solve to obtain $y_1 = \tilde{L}^{-1} r$;
  - multiplication $y_2 = W^T y_1$;
  - forward and back substitution with L_d to obtain y_3 from $L_d L_d^T y_3 = y_2$;
  - multiplication with W to obtain $y_4 = y_1 + W^T y_3$;
  - back substitution with $\tilde{L}^T$ to obtain $y = \tilde{L}^{-T} y_4$.

This reorganization of the SMW formula has, in contrast to the straightforward implementation with taking W as a solution of $\tilde{L}\tilde{L}^T W = A_d D_d$, the advantage of saving one backward solve with $\tilde{L}^T$. However, we need the ability to perform forward and backward solves with the $\tilde{L}$ factor independently, rather than having to perform both operations jointly.

4 Computational Experience

Our implementation of the dense column handling strategy was based on the beta-2.0 release (October 1996) of PCx [3], a primal-dual interior-point code that implements Mehrotra's [9] predictor-corrector algorithm for linear programming. (The modifications were subsequently incorporated into release 1.0 of PCx, dated March 1997.) The sparse Cholesky routine is from the code of Ng and Peyton [10], release 0.4 (May 1995), which implements a multiple minimum degree ordering strategy. A small modification to the Cholesky algorithm is needed to handle small pivots: If a pivot is identified as being too small (or negative), it is replaced by 10^{128}, which has the effect of inserting a zero component into the solution vector at the appropriate location. This modification is well established for interior-point codes in various contexts; see Wright [12] for a theoretical investigation. We report results on a Sun SPARCstation 20 running SunOS 4.1.4.

In the current NETLIB set only eight problems contain dense columns. We consider five of these problems to be large and the others to be small. Even though it is not usually necessary to extract dense columns for the small examples, since their runtimes are so short in any case, we note that algorithms for solving instances of stochastic linear programming may make multiple calls to LP solvers, so even a small savings in runtime can be significant in these cases.

Figures 1 and 2 show all problems from the NETLIB collection having dense columns. The constraint matrices are plotted on the left-hand side, while the right-hand side shows the first few columns of A, ordered by the number of nonzeros (vertical axis) and plotting the number of nonzeros in each column (horizontal axis). Note the logarithmic scale on the vertical axis.

Our experience showed that the SMW approach described above was often numerically unstable. We explain this fact as follows: Because of the structural role which the dense columns play in the real-world models defining the linear programs, some of them are almost always included in an optimal basis. Hence, the matrix $A_s D_s^2 A_s^T$ may approach singularity near the solution, and $W = \tilde{L}^{-1} A_d$ may have only a few digits of accuracy in its smallest elements. Even these digits may be lost in forming $D_d^{-2} + W^T W$, and so the solutions calculated by SMW are often inaccurate. Lustig, Marsten, and Shanno [8] state a example in three dimensions where $A_s D_s^2 A_s^T$ approaches rank deficiency as the interior-point method approaches the optimum.

The numerical experience shows that the residual $b - AD^2A^Tx$ of the normal equations that we are solving tends to increase during the last stages of the interior-point algorithm. Whenever this occurs, the infeasibility of the iterates increases dramatically, causing the interior-point method to break down.

Hence, we use a pcg algorithm (see Golub and Van Loan [6, algorithm 10.3.1, p. 529]) to refine the solutions. The sparse part $A_s D_s^2 A_s^T$ is used as a preconditioner; no additional work is needed to compute it since its Cholesky factorization is known already. The pcg technique achieved the desired level of accuracy in the search directions quite well. However, in cases in which the diagonal modification technique was used to replace small pivots, the pcg method often failed. This result is not surprising; the preconditioner is essentially singular in this case, and the large-element substitution technique ensures that the corresponding components of the pcg modifications are fixed at zero, so no improvement in these components can occur.

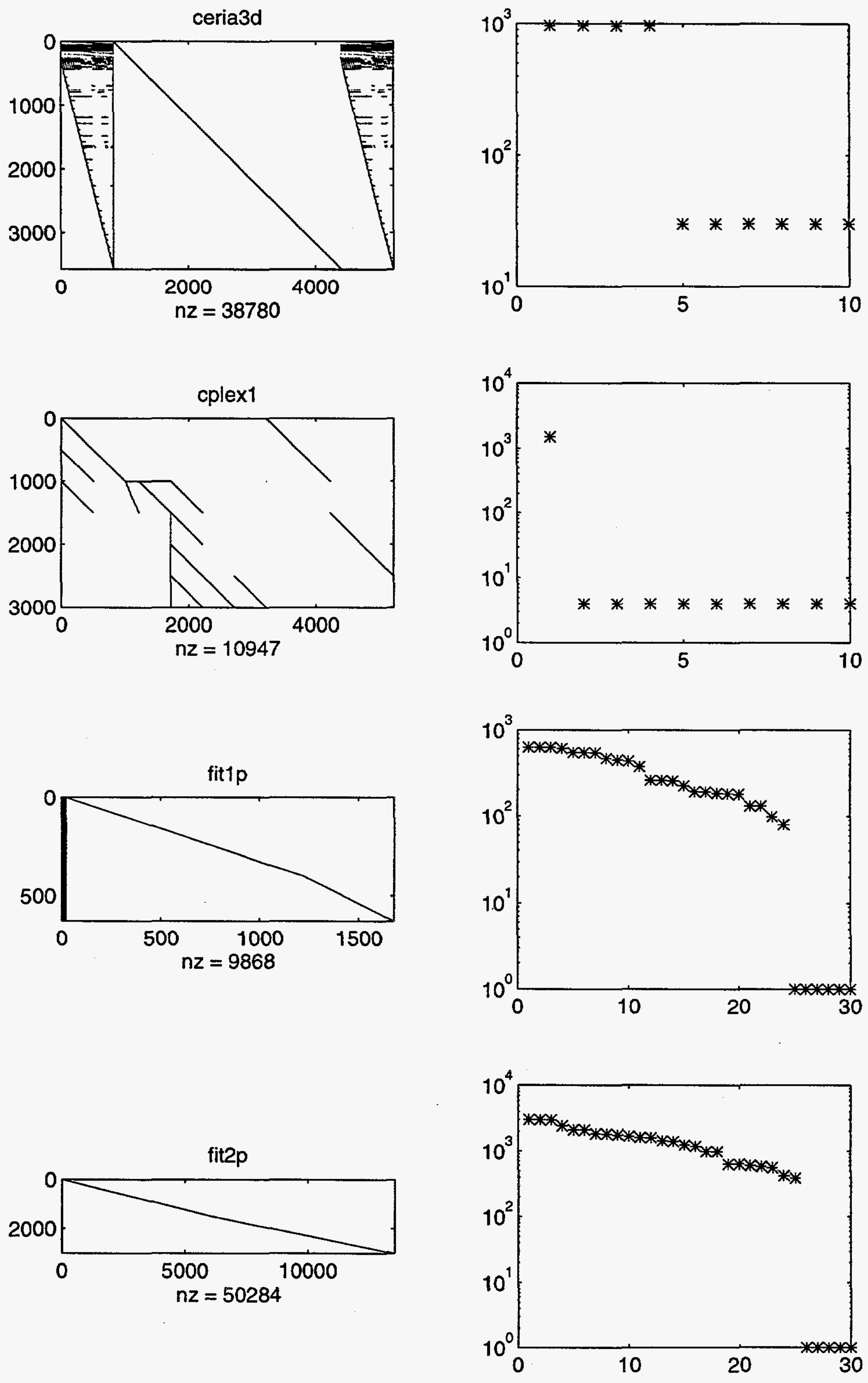

Figure 1: Problems from NETLIB with dense columns (I)

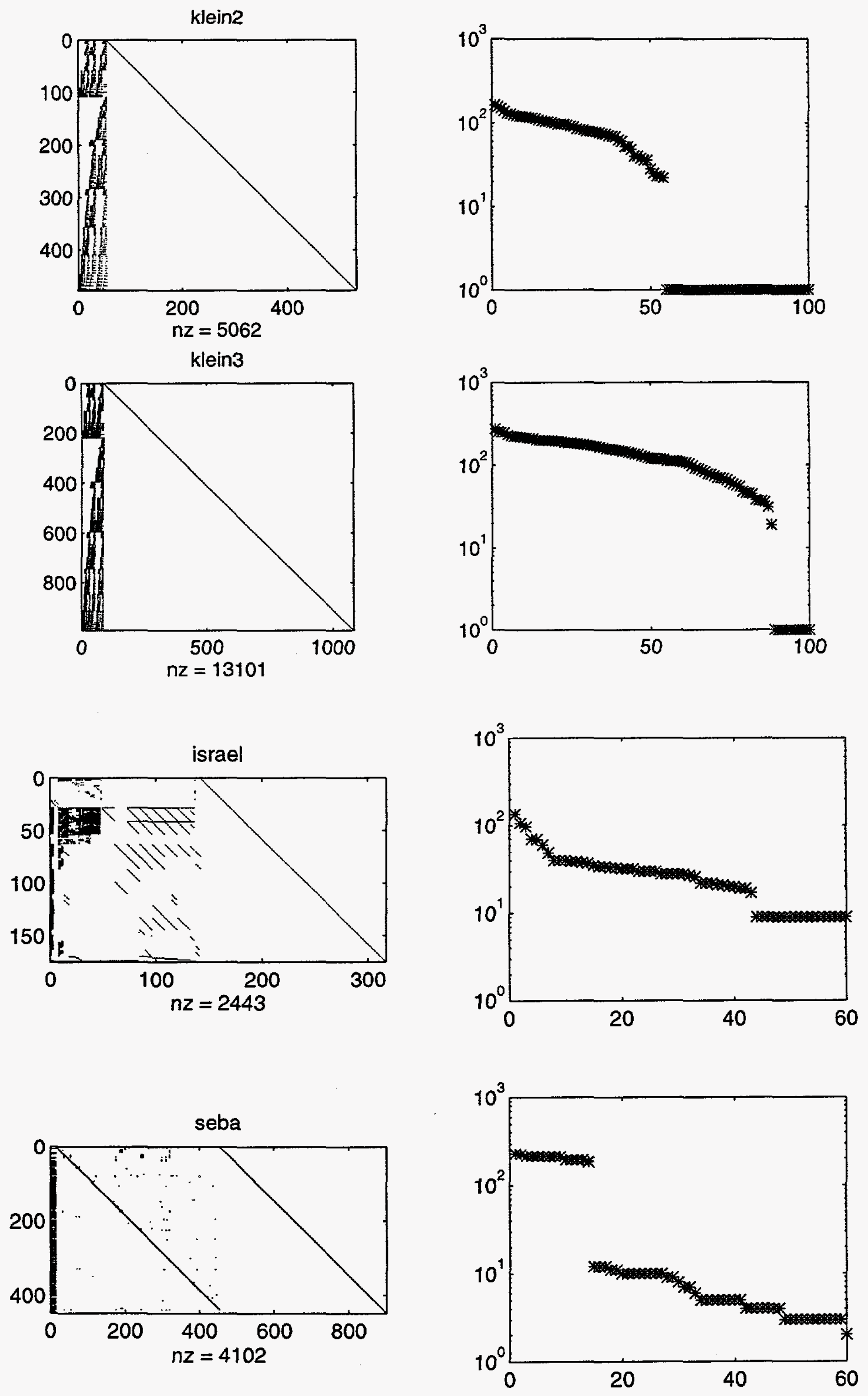

Figure 2: Problems from NETLIB with dense columns (II)

Our combined SMW-pcg approach can be outlined as follows:

- Flag all columns with column density greater than ρ as dense, where ρ depends on the number of rows m in the (preprocessed) A as follows:

$$\rho = \begin{cases} 1.0 & \text{for } m \leq 500 \\ 0.2 & \text{for } 500 < m \leq 1000 \\ 0.1 & \text{for } 1000 < m \leq 2000 \\ 0.05 & \text{for } m > 2000. \end{cases}$$

- Perform a modified Cholesky-factorization of $A_s D_s^2 A_s^T$ and set the flag `dopcg` to `false` if small diagonals are present.
- Apply the SMW formula as described above to solve the linear system.
- If this solution does not yield a sufficiently small relative residual for the normal equations, and `dopcg` is `true`, then enter the pcg routine.
- Exit pcg when either the relative residual has decreased sufficiently or the number of pcg iterations exceeds $10 \cdot n_d$. In the latter case, restore the original solution obtained from the SMW formula if the relative residual has not been improved by the pcg procedure.

A similar technique is used by the LIPSOL code of Zhang [14, 15].

The pcg procedure need not be used only in conjunction with SMW. It can be used in place of iterative refinement to improve the accuracy of the solutions even when dense columns are absent. In this case, the preconditioner is simply the computed factorization of AD^2A^T and the maximum number of pcg iterations is 10. In general, pcg yields better results than iterative refinement for the same number (or fewer) of improvement iterations.

Table 1 shows the effect of extracting dense columns, when we choose ρ not by the strategy above but rather manually, to achieve maximum efficiency. We tabulate the dimensions of the problem, the number of dense columns, the densities of AA^T (which is the same as the density of AD^2A^T) and $A_sA_s^T$, and the densities of the Cholesky factors of these matrices. Note that we could not obtain a solution of the full AD^2A^T system for the problem `fit2p` in a reasonable amount of time, so the corresponding entry of the table is missing.

We stress that only the `klein` problems were sensitive to the choice of the threshold parameter ρ; in all other instances there is sharp distinction between the dense and sparse columns. (In the problem `israel`, there is a relatively dense squared window of 27 columns, but extraction of this window only halves the density and more than doubles the CPU time when compared with the situation displayed above.)

It is important to remark, in the context of Section 2, that all feasible problems could be solved to the desired accuracy (a relative error of 10^{-8} in primal infeasibility, dual infeasibility, and duality gap). Furthermore, there is no change in the number of interior-point iterations needed, by comparison with the case in which no dense columns are extracted. Note that PCx terminates with optimal status for `fit1p`, `fit2p`, `israel`, and `seba`, with infeasible status for `ceria3d` and `cplex1`, and with unknown status for `klein2` and `klein3`. For reference we mention that `fit1p`, `fit2p`, `israel`, and `seba` are feasible problems, while the other four are infeasible problems.

A remark is in order concerning the problems `klein2` and `klein3`, which are infeasible but for which PCx terminates with status "unknown". We extracted as many columns as possible, with the result that the preconditioner is so poor that pcg does not make any progress and is aborted after $10n_d$ iterations. This behavior happens only in the last stages of the interior-point algorithm. According to our explanation above, we observe that when extracting only few columns, pcg converges in both

Table 1: Computational results: Maximum efficiency

	ceria3d	cplex1	fit1p	fit2p	klein3
number of columns	4400	5224	1677	13525	1082
number of rows	3576	3005	627	3000	994
number of dense columns	4	1	24	25	88
density of AA^T	0.154	0.251	1.000	1.000	0.564
density of $A_sA_s^T$	0.0105	0.0019	0.0016	0.0003	0.0010
density, Cholesky factor of AA^T	0.247	0.252	1.000	1.000	0.679
density, Cholesky factor of $A_sA_s^T$	0.0113	0.0020	0.0016	0.0003	0.0010
solution time using A [s]	3273	494	250	?	480
solution time using A_s [s]	28.6	1.9	6.9	51.6	373
% reduction in comp–time	99.1%	99.6%	97.2%	~100%	22.3%

	israel	seba	klein2
number of columns	316	901	531
number of rows	174	448	477
number of dense columns	15	14	54
density of AA^T	0.735	0.510	0.610
density of $A_sA_s^T$	0.131	0.007	0.002
density, Cholesky factor of AA^T	0.753	0.533	0.699
density, Cholesky factor of $A_sA_s^T$	0.0074	0.133	0.002
solution time using A [s]	4.7	27	55
solution time using A_s [s]	2.4	1.8	105

problems in a moderate number of iterations. In the problems `klein2` and `klein3` the relatively high computation time (when compared with the number of nonzeros in the remaining matrix) is due to the bad performance of pcg and the large number of iterations allowed. The number of interior-point iterations needed by the algorithm coincides with the case of unmodified A.

When we used the heuristic outlined above to choose ρ, the results of Table 2 were obtained. Four of the problems were solved with a similar level of efficiency to the best possible level, but the remaining one—`klein3`—showed a degradation in efficiency even when compared with the case in which no dense columns are extracted. The slower performance was caused by the need for up to 65 pcg iterations at each step. Still, this problem converged in a reasonable time, as did the others.

Table 2: Computational results for the default strategy

	ceria3d	cplex1	fit1p	fit2p	klein3
number of columns	4400	5224	1677	13525	1082
number of rows	3576	3005	627	3000	994
number of dense columns	4	1	22	25	17
density, AA^T	0.154	0.251	1.000	1.000	0.564
density, $A_sA_s^T$	0.0105	0.0019	0.0423	0.0003	0.4451
density, Cholesky factor of AA^T	0.247	0.252	1.000	1.000	0.679
density, Cholesky factor of $A_sA_s^T$	0.0113	0.0020	0.0423	0.0003	0.597
solution-time using A [s]	3273	494	250	?	480
solution-time using A_s [s]	28.6	1.9	9.7	51.6	686
% reduction in comp–time	99.1%	99.6%	96.1%	~100%	-42.9%

Figure 3 shows the efficiency of the pcg refinement process. Each plot shows the number of pcg iterations (vertical axis) needed to converge to a relative accuracy of 10^{-8} in the residual, versus the interior-point iteration counter (horizontal axis). By comparing the number of dense columns extracted (stated in the head of each plot) with the number of pcg-iterations, one can see that both correlate nicely at some problems, but on others pcg took much more iterations than n_d (the number of columns extracted). In the absence of rounding errors n_d iterations are enough to correct the rank-n_d perturbation of the Sherman-Morrison-Woodbury formula. For the large problems (upper half of the figure) we provide the data for the same settings as we took for Table 2. (We omit `cplex1`, which terminates after 3 ipm iterations without needing any pcg refinement.) For the small problems, we used the specifications of Table 1. For **klein2**, we provided in addition the results for extracting only 23 columns, where convergence of the refinement process still could be achieved. As for the case of extracting 54 columns, we stopped pcg after 540 iterations, according to the heuristic described above.

Figure 4 presents some representative plots of the relative residual in the progress of the preconditioned conjugate gradient method. In the head of each plot we list the iteration number of the interior-point method and, after the slash, the total number of interior-point iterations needed to solve the problem. In addition, the number of columns extracted may serve as a clue to the efficiency of the refinement procedure. While the horizontal axes give the number of the pcg-iteration, the logarithmically vertical axes represent the relative residual $\frac{\|b-AD^2A^Tx\|}{\|x\|}$ of the solution x. The predictor step is plotted as a continuous line, while the corrector step is plotted as a dashed line. Yet again, we used the default strategy as described in Table 2. While **israel**, **fit1p**, and **fit2p** show excellent behavior, the infeasible problem **klein2** and its close relative **klein3** typically show an increase in the residual before eventually bringing it below the goal of 10^{-8}.

Figures 3 and 4, along with additional data not presented here, indicate that it is more difficult to solve for the corrector search direction than for the predictor search direction. This result suggests that the optimal number of corrections in higher-order predictor-corrector methods is one, as many authors have previously noted.

5 Conclusions

We conclude by pointing to some theoretical and practical aspects that require further attention.

Additional processing can be applied to the reduced matrix A_s to ensure that structural nonsingularity is not present (for example, to eliminate empty rows).

The determination of density threshold ρ can be improved by making it more adaptive. For instance, we could look for a sizeable gap in the density profile, as plotted in the graphs of Figures 1 and 2. We could also try a strategy based on trying different values of the threshold and evaluating their effect on the densities of $A_sA_s^T$ and of the Cholesky factor of this matrix. Some estimate of the cost per iteration associated with each value could then be made, and possibly adjusted after a few steps when some information on the required number of pcg iterations is gathered, and the "best" value of ρ could be chosen accordingly.

A thorough analysis of the numerical effects of extracting dense columns has yet to be performed, to our knowledge. As we mentioned, it is quite conceivable that AD^2A^T could be approaching a well-conditioned limit while its reduced form $A_sD_s^2A_s^T$ approaches an ill-conditioned limit, making the SMW solution procedure potentially unstable.

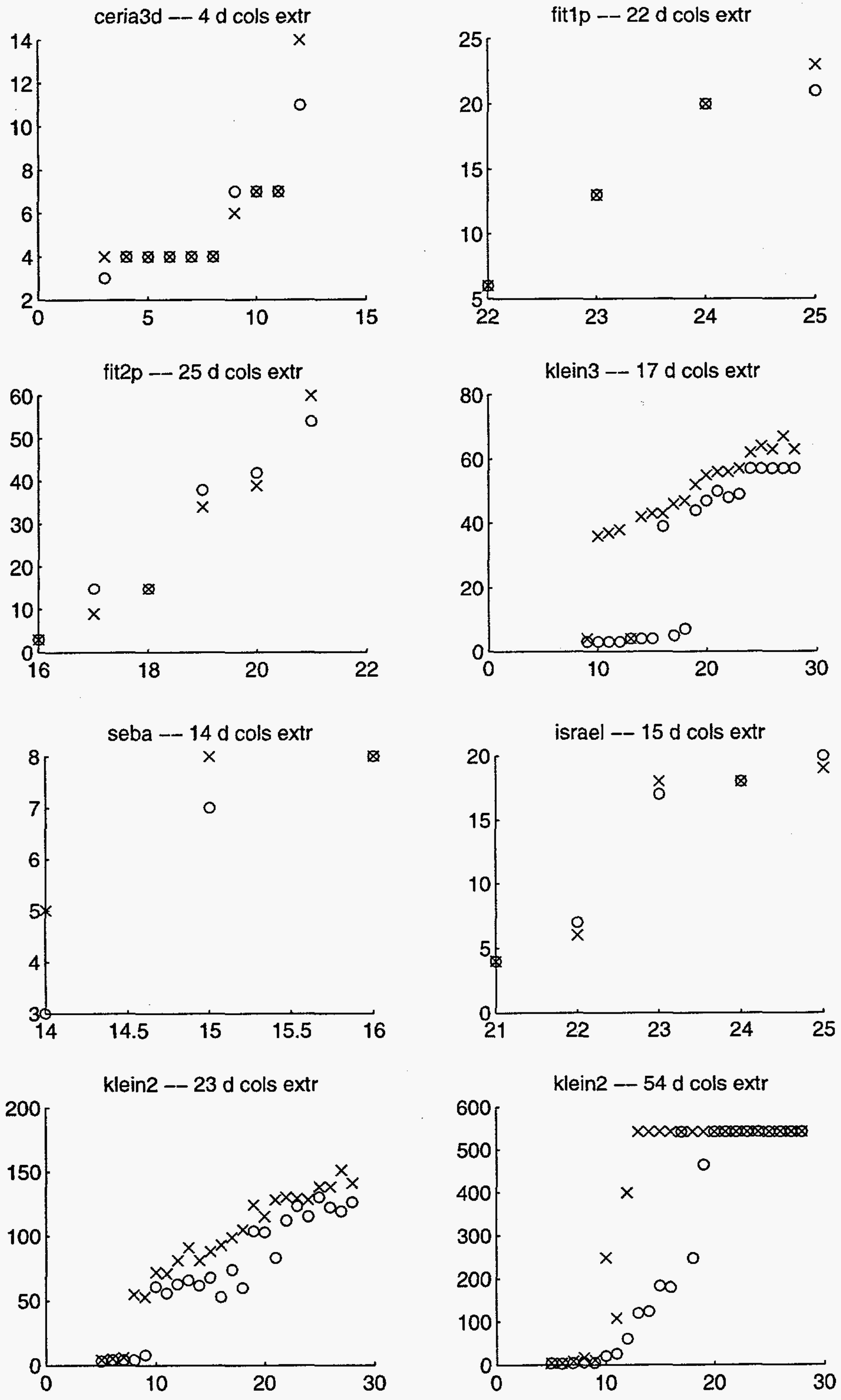

Figure 3: The number of pcg-iterations in each ipm-step (predictor [○] and corrector [×])

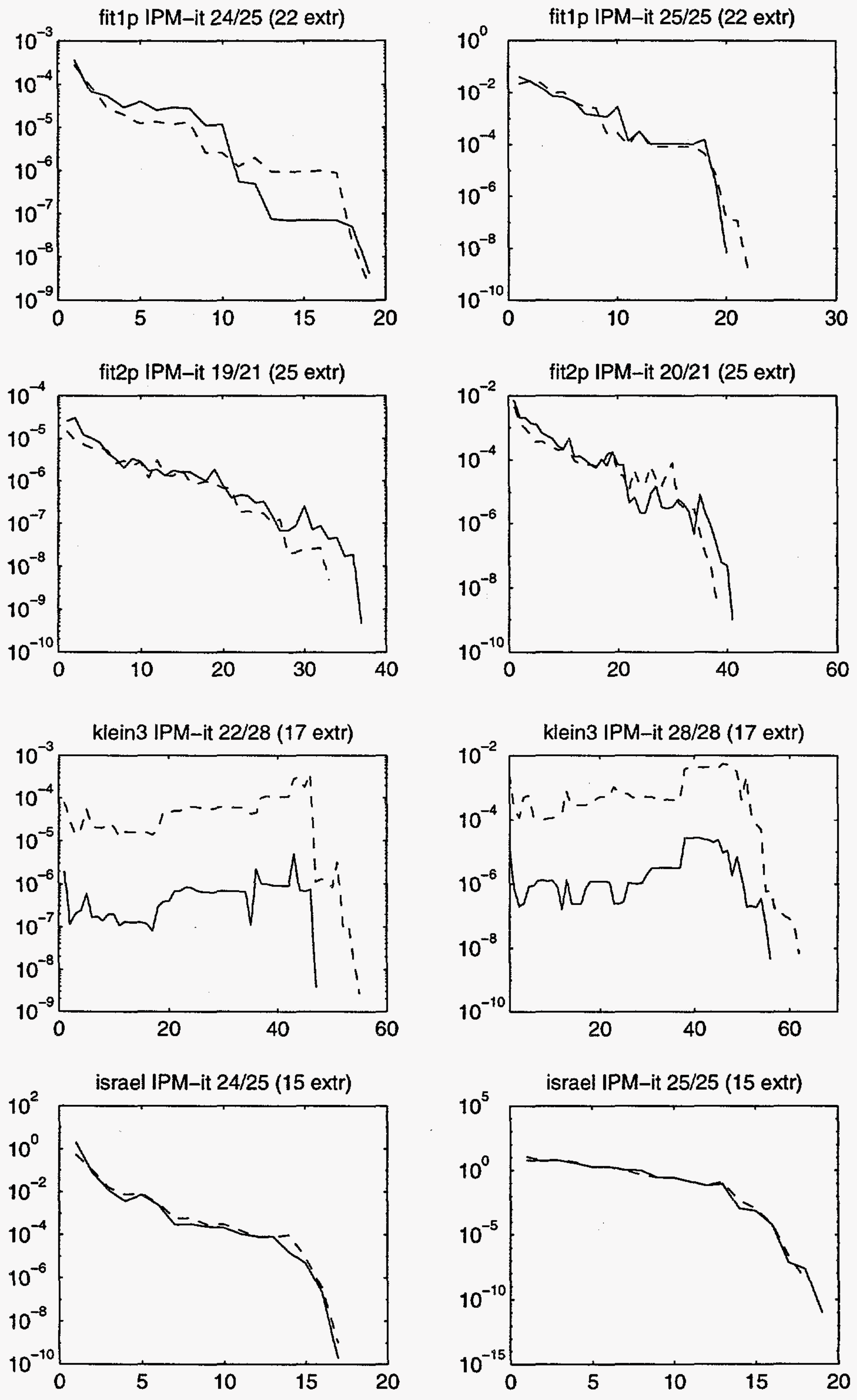

Figure 4: The relative residual of the solution, as decreased by pcg: predictor step=continuous line, corrector step=dashed line.

Acknowledgments

The first author is grateful to Florian Jarre for his advice and support, and to Argonne National Laboratory for supporting his visit during the fall of 1996.

References

[1] Ilan Adler, Narenda Karmakar, Mauricio G. C. Resende, and Geraldo Veiga. An implementation of Karmakar's algorithm for linear programming. *Mathematical Programming*, pages 297–335, 1989. Errata in Mathematical Programming 50:415, 1991.

[2] Chan Choi, Clyde L. Monma, and David L. Shanno. Further development of a primal-dual interior point method. *ORSA Journal on Computing*, 2(4):304–311, Fall 1990.

[3] Joseph Czyzyk, Sanjay Mehrotra, and Stephen J. Wright. *PCx User Guide*. Optimization Technology Center, Argonne National Laboratory and Northwestern University, Mathematics and Computer Science Division, Argonne National Laboratory 9700 South Cass Avenue, Argonne, IL 60439, October 1996.

[4] Robert Fourer and Sanjay Mehrotra. Solving symmetric indefinite systems in an interior-point method for linear programming. *Mathematical Programming*, 62:15–39, 1993.

[5] Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright. On projected Newton methods for linear programming and equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[7] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and Its Applications*, 152:191–222, 1991.

[8] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. On implementing Mehrotra's predictor-corrector interior point method for linear programming. *SIAM Journal on Optimization*, 2(3):435–449, August 1992.

[9] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, November 1992.

[10] E. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal on Scientific Computing*, 14:1034–1056, 1993.

[11] Robert J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra and Its Applications*, 152:107–117, 1991.

[12] Stephen Wright. Modified Cholesky factorizations in interior-point algorithms for linear programming. Preprint ANL/MCS-P600-0596, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, May 1996.

[13] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM Publications, Philadelphia, PA, December 1996.

[14] Yin Zhang. *User's Guide to LIPSOL*. Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Maryland 21228-5398, U.S.A., version 0.3 edition, July 1995. Documentation for the package LIPSOL.

[15] Yin Zhang. Solving large-scale linear programs by interior-point methods under the MATLAB enviroment. Technical Report TR96-01, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Maryland 21228-5398, U.S.A., February 1996. Documentation for the package LIPSOL.