CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université de Paris-Dauphine)
Unité Associée au CNRS n^0 825

ON HANDLING DENSE COLUMNS OF CONSTRAINT MATRIX IN INTERIOR POINT METHODS OF LARGE SCALE LINEAR PROGRAMMING [1]

CAHIER N^0 105

J. GONDZIO [2]

avril 1991

Received : October 1990.

---

[1] *The results discussed in this paper have been obtained when the author was staying at LAMSADE. A preliminary version of the paper has been presented at the Applied Mathematical Programming and Modelling Symposium APMOD '91 in London, January 14-16, 1991.*

[2] *Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-47 Warsaw, Poland.*

CONTENTS

Une méthode de traitement des colonnes pleines pour des programmes linéaires de grande taille dans les méthodes intérieures

RESUME

Dans le cadre des méthodes intérieures, une matrice de la forme AA^T doit être inversée à chaque itération. Si la matrice A contient des colonnes pleines, celles-ci, lors du produit AA^T, créent des sous-matrices pleines. Le facteur de Cholesky devient alors non-creux et l'efficacité du code linéaire décroît. Nous proposons donc, dans cet article, une méthode de traitement des colonnes pleines de programmes linéaires de grande taille. Elle consiste à remplacer chaque colonne pleine par un ensemble de colonnes ayant moins d'éléments non nuls. Ainsi, la densité de la matrice AA^T et, par conséquent, celle du facteur de Cholesky, s'en trouvent diminuées.

Mots-clés: Projections de Karmarkar, Programmation Linéaire, Colonnes Pleines

On handling dense columns of constraint matrix in interior point methods of large scale linear programming

ABSTRACT

A method is proposed for handling dense columns of the constraint matrix of large scale linear programming problems. Such columns are known to create dense windows in matrix AA^T that has to be inverted at every iteration of the interior point method. Consequently, Cholesky factor of AA^T becomes dense, which degrades the efficiency of the LP code. In the method considered, all dense columns are then split into shorter ones. One dense AA^T window of large size is thus replaced with p windows each of size p times smaller, leading to a remarkable reduction of the number of nonzeros in the matrix to be inverted and in its Cholesky factor.

Key words: Karmarkar's Projections, Linear Programming, Dense Columns.

1. Introduction

We are concerned with solving linear programming problem

$$\text{minimize} \quad c^T y \,, \tag{1a}$$

$$\text{subject to} \quad Ay = b \,, \tag{1b}$$

$$y \geq 0 \,, \tag{1c}$$

where $A \in \mathbb{R}^{m \times n}$, $c, y \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. We assume that an interior point method derived from a logarithmic barrier approach is applied to it. Algorithmic techniques however are not discussed in this paper (the reader interested in them is referred to papers of Adler et al. (1989b), Gill et al. (1986), Choi et al. (1990) and other references therein).

Instead, a method is proposed for solving equation

$$AA^T x = d, \tag{2}$$

where $AA^T \in \mathbb{R}^{m \times m}$ and $x, d \in \mathbb{R}^m$, which is crucial for the efficiency of the whole LP code. Let us mention that the equation that have to be solved at every iteration of the method has in general slightly different form from (2). For ease of presentation however, we have omitted matrix D in it and the fact that the matrix A that appear in (2) may differ from the constraint matrix (1b) with a bordered rows.

Although the problem constraint matrix A is usually very sparse, matrix AA^T in equation (2) may be quite dense. Any column of matrix A that has k nonzero entries causes creating dense window of size $k \times k$ in matrix AA^T (subject to its symmetric permutation). Standard approaches to solve (2) such as applying QR factorization to matrix A^T or computing Cholesky decomposition of matrix AA^T (see e.g., chapters 6 and 5 of the book of Golub and Van Loan (1983), respectively) may then be completely inefficient, especially if k is comparable with m. Matrix R or Cholesky factor L contain in such case triangular dense window of dimension at least k (its size is often larger due to the *fill-in* i.e. new nonzero elements added

during the factorization process).

For the above reasons several methods for 'special treatment' of dense columns in the constraint matrix have been proposed. The first one is to avoid formulating AA^T explicitly and to apply iterative procedure of conjugate gradient (see e.g., Golub and Van Loan 1983, chapter 10) to solve (2). A sparse preconditioner $L_s L_s^T = A_s A_s^T$, where A_s is a sparse part of A, i.e. it is matrix A with dense columns removed, is used to accelerate the convergence and improve numerical properties of the method (see e.g., Munksgaard 1980). This approach is due to Gill et al. (1986) and was implemented for example by Adler et al. (1989a). Another approach, due to Gill et al. (1988), is to handle dense columns in the form of the Schur complement and to apply Sherman-Morrison-Woodbury formula (see e.g., Hager 1989) to solve equation (2). It is not iterative in character and when carefully implemented (see e.g., Lustig et al. 1990) it proved to be very efficient.

However, dense columns are not so often present in real-life linear programs (in extended *Netlib* collection of Gay (1985) only 4 of all 86 problems contain columns dense enough to motivate applying special techniques to handle them). A need then arise for rather easily implementable method that does not add significant overhead to the LP code but well prevents degrading influence of dense columns on the speed of computing projections.

In this paper such a method is proposed. A procedure for preprocessing the linear program is suggested which ends up with creating equivalent problem that has better structured constraint matrix $\underline{A}$. Consequently, the number of nonzero elements of matrix $\underline{A}\underline{A}^T$ is remarkably smaller than that of AA^T. A significant reduction of nonzero counts of Cholesky factor of $\underline{A}\underline{A}^T$ can thus be obtained.

To achieve this, any variable associated with dense column of length k is replicated and the dense column is cut into pieces. Variable y_i is replaced by p variables $y_{i1}, y_{i2}, \ldots, y_{ip}$ each of these being associated

with appropriate part of the long column that has only k/p nonzero elements. Summing up, instead of k^2 nonzero elements in AA^T we have $p(k/p)^2 = k^2/p$ nonzero elements in p small dense windows of AA^T (we neglect terms linear on k). Additionally, $p-1$ new constraints of type

$$y_{ij} - y_{ij+1} = 0 , \quad j = 1,2,\ldots,p-1, \tag{3}$$

are bordered to a linear program.

Let us observe that the method proposed is particularly easy to implement and that it can be included into the preprocessing of a linear program. Consequently, the overhead added by it to the LP code is negligible.

The method is derived from techniques used widely in multistage stochastic linear programming, where the variables of earlier stages are sometimes replicated to allow formulating all the scenarios independently (see e.g., Rockafellar and Wets 1987 and Lustig et al. 1989) and obtain easily decomposable structures. Mulvey and Ruszczynski (1990) indicate advantages of applying such approach to two specially structured linear programming problems: multistage stochastic and multicommodity network ones. Additionally, since in these particular problems the number of linking constraints of type (3) may be very large, they suggest applying augmented Lagrangian to handle them.

In this paper general large scale linear programming problems are addressed. The method presented is an easily implementable remedy for dense columns in the constraint matrix. It has been experimentally implemented and compared with a direct approach i.e. Cholesky factorization of AA^T. Preliminary computational results proved that the application of the method of splitting significantly accelerates computation of orthogonal projections.

The paper is organized as follows. In Section 2 a motivation for splitting dense columns is given. In section 3 a detailed description of the method is made and in Section 4 issues of implementation are analysed.

Finally, Sections 5 and 6 bring numerical results and conclusions, respectively.

2. Dense columns in linear programs

We start this section from showing the degrading influence of the existence of a single dense column in the LP constraint matrix on the efficiency of an interior point method of linear programming. The following example illustrates the problem. Let us suppose that m = 8, n = 9 and the sparsity pattern of a constraint matrix has the form

$$A = \begin{bmatrix} \times & \times & & & & & & \\ \times & & \times & & & & & \\ \times & & & \times & & & & \\ \times & & & & \times & & & \\ \times & & & & & \times & & \\ \times & & & & & & \times & \\ \times & & & & & & & \times \end{bmatrix}, \quad AA^T = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \end{bmatrix} \quad \text{and} \quad L = \begin{bmatrix} d & & & & & & & \\ \times & d & & & & & & \\ \times & \times & d & & & & & \\ \times & \times & \times & d & & & & \\ \times & \times & \times & \times & d & & & \\ \times & \times & \times & \times & \times & d & & \\ \times & \times & \times & \times & \times & \times & d & \\ \times & \times & \times & \times & \times & \times & \times & d \end{bmatrix}, \tag{4}$$

so AA^T is an 8×8 full matrix with 28 elements below the diagonal that have to be stored. The Cholesky factor L has also 28 entries below the diagonal. In other words, full matrix technology has to be applied to solve this sparse problem (let us observe that every simplex basis matrix of the problem is triangular and sparse).

Let us now introduce variables y_{11}, y_{12} (both equal to y_1), split the first column of A into two equal parts and add a constraint of type (3) that ties up replicated variables. The new constraint matrix has now the form

$$\underline{A} = \begin{bmatrix} \times & & & & & & & & \\ \times & \times & & & & & & & \\ \times & & \times & & & & & & \\ \times & & & \times & & & & & \\ & \times & & & \times & & & & \\ & \times & & & & \times & & & \\ & \times & & & & & \times & & \\ & \times & & & & & & \times & \\ 1 & -1 & & & & & & & \end{bmatrix}, \quad \underline{AA}^T = \begin{bmatrix} \times & \times & \times & \times & & & & & \times \\ \times & \times & \times & \times & & & & & \times \\ \times & \times & \times & \times & & & & & \times \\ \times & \times & \times & \times & & & & & \times \\ & & & & \times & \times & \times & \times & \\ & & & & \times & \times & \times & \times & \\ & & & & \times & \times & \times & \times & \\ & & & & \times & \times & \times & \times & \\ \times & \times & \times & \times & & & & & 2 \end{bmatrix} \quad \text{and} \quad \underline{L} = \begin{bmatrix} d & & & & & & & & \\ \times & d & & & & & & & \\ \times & \times & d & & & & & & \\ \times & \times & \times & d & & & & & \\ & & & & d & & & & \\ & & & & \times & d & & & \\ & & & & \times & \times & d & & \\ & & & & \times & \times & \times & d & \\ \times & \times & \times & \times & \times & \times & \times & \times & d \end{bmatrix}, \tag{5}$$

so $\underline{AA}^T$ is a 9×9 matrix with only 20 elements below the diagonal and so is its Cholesky factor.

Consequently, instead of solving linear program with constraint matrix (4) we may solve better structured problem with matrix (5) and (at every

iteration of the interior point method) take advantage of the lower density of Cholesky factor $\underline{L}$ of $\underline{A}\underline{A}^T$.

3. Some theory of splitting

We show in this section how the splitting can be done in a given linear program to obtain equivalent problem that has better sparsity pattern, i.e. leading to sparser Cholesky factor of matrix $\underline{A}\underline{A}^T$.

Let us consider a linear programming problem of the form (1). To simplify the presentation we assume that the first column of its constraint matrix is split into two parts. We thus introduce partition

$$A = \left[\begin{array}{c|c} a_1 & A_0 \end{array}\right] . \tag{6}$$

and define an $(m+1)\times(n+1)$ matrix

$$\underline{A} = \left[\begin{array}{c|c|c} \underline{a}_{11} & \underline{a}_{12} & A_0 \\ \hline 1 & -1 & 0 \end{array}\right] \tag{7}$$

which together with vectors

$$\underline{b} = (b^T,0)^T \in \mathbb{R}^{m+1}, \tag{8a}$$

$$\underline{c} = (c_1,0,c_2,\ldots,c_n)^T \in \mathbb{R}^{n+1}, \tag{8b}$$

$$\underline{y} = (y_{11},y_{12},y_2,\ldots,y_n)^T \in \mathbb{R}^{n+1} \tag{8c}$$

determine new LP problem

$$\text{minimize} \quad \underline{c}^T\underline{y} , \tag{9a}$$

$$\text{subject to} \quad \underline{A}\underline{y} = \underline{b} , \tag{9b}$$

$$\underline{y} \geq 0 , \tag{9c}$$

Let us suppose that

$$\underline{a}_{11}+\underline{a}_{12} = a_1 . \tag{10}$$

Observe that y is a feasible point of (1) if and only if $\underline{y}$ defined by

$$y_1 = y_{11} = y_{12} \tag{11a}$$

$$y_j = y_j \ , \quad j = 2,3,\ldots,n \ . \tag{11b}$$

is a feasible point of (9).

Proposition 3.1. Optimal solution of (1) exists if and only if optimal solution of (9) exists and if they both exist, then they satisfy (11).

We omit straightforward proof of this result and pass to discuss its practical significance.

Any dense column of the constraint matrix containing, say, k nonzero entries can be split into two parts (equation (10)) containing q and $k-q$ elements, respectively. Thus, instead of a dense window of size $k\times k$ in AA^T we obtain two dense windows of sizes $(q+1)\times(q+1)$ and $(k-q+1)\times(k-q+1)$, respectively. The best possible reduction of nonzero elements in $\underline{AA}^T$ is obviously obtained when $q = k/2$.

Let us observe that part a_{12} of (7) can be further split into two shorter parts (a new linking constraint will have to be bordered to a constraint matrix then). It is equivalent to splitting the original column a_1 of (6) into three parts. More generally (as follows from proposition 3.1 by induction), any dense column of length k can be split into p equal parts so the dense $k\times k$ window in AA^T may be replaced with p dense windows each of dimension $k/p+1$ or $k/p+2$ in $\underline{AA}^T$ matrix. If we neglect linear terms and omit completely the influence of the rest of the constraint matrix (submatrix A_0 in equality (6)) on the sparsity structure of $\underline{AA}^T$, then we may conclude that instead of k^2 elements in AA^T we now deal with k^2/p nonzero entries in $\underline{AA}^T$. Such substantial reduction of the number of entries in the matrix $\underline{AA}^T$ is supposed to lead to a remarkable savings of space required by its Cholesky factor and to a significant acceleration of both symbolic and numerical phases of the decomposition of $\underline{AA}^T$.

Another advantage of the approach presented is that splitting itself does not cause nonsingularity of $\underline{A}\underline{A}^T$.

Proposition 3.2. Let A and $\underline{A}$ be defined by (6) and (7), respectively. If (10) holds and $\mathrm{rank}(A) = m$, then $\mathrm{rank}(\underline{A}) = m + 1$.

Proof. The proof follows easily the observation that replacing the second column of (7) by the sum of its two first columns we obtain

$$\begin{array}{|c|c|c|} \hline \underline{a}_{11} & a_1 & A_0 \\ \hline 1 & & 0 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|} \hline \underline{a}_{11} & A \\ \hline 1 & 0 \\ \hline \end{array} \quad . \qquad \blacksquare$$

Let us mention that removing dense columns that takes place when a preconditioner for conjugate gradient method is defined or Schur complement approach is applied, may cause serious stability problems. If $\mathrm{rank}(A_0)$ is less than m, which is often the case since the removed dense column is usually an integral part of the problem formulation, then $A_0A_0^T$ becomes singular. The Schur complement approach would fail in such a situation while the conjugate gradient method might still work under the condition that the rank deficiency of $A_0A_0^T$ is removed to allow computing the preconditioner (see e.g., Munksgaard 1980). To achieve this, a positive constant is usually added to too small diagonal elements of $A_0A_0^T$ when its Cholesky factor is computed or columns that have too small diagonal elements are simply replaced with appropriate unit vectors (see e.g., Adler et al. 1989b). It leads however to less accurate preconditioner, which may slow down the convergence of the method. It may seem that introducing artificial variables should ensure nonsingularity of $A_0A_0^T$ allowing safe application of Schur complement method. Computational practice of Lustig et al. (1990) did not however confirm this since artificial variables tend to zero when optimum is approached and the instability still manifests itself (an automatical switch

to preconditioned conjugate gradient method is suggested in such case).

In the approach presented in this paper the analogous difficulties are less probable (proposition 3.2. ensures that $\underline{A}\underline{A}^T$ is always nonsingular). If they anyway occur, then there still exists the possibility of switching to preconditioned conjugate gradient method.

4. Implementation

The method of splitting dense columns of the constraint matrix has been experimentally implemented. It is advantageous that the analysis of the sparsity pattern of A have to be done only once and that it can be completed before the solution of the problem starts. It thus can be included to preprocessing of the LP coefficient matrix.

Let us mention however that it is not the only way of implementing it. Vanderbei (1990) suggests that it may be preferable to solve the problem in the same form as stated originally and use splitting technique implicitly in a module that solves equation (2). Advantage can then be taken of the smaller size of the problem since the linking constraints of type (3) are not bordered to it. On the other hand, the analysis that is necessary to determine optimal splitting must be repeated in such case at every iteration of the interior point method, which leads to an unnecessary increase of the computation time. Vanderbei (1990) selects the simplest possible splitting (every column containing more than k nonzeros, where k is a given threshold value, is split into a set of columns, each containing exactly k nonzeros except for the last column, which contains the remainder of entries and the nonzeros are allocated to the new columns in the order in which they appear in an internal data structure). The cost of splitting in his method is thus negligible while in the method presented in this paper an advanced analysis of the sparsity pattern of A is made to determine optimal splitting that ensures minimum storage requirements and maximum acceleration of the

Cholesky factorization. Although the time of splitting in our implementation never exceeds 15% of the time required by the reordering and the symbolic factorization, we want to avoid adding it at every iteration.

Let us now pass to the description of a heuristic that has been applied to determine the partition of a long column. First, let us analyse in more detail the consequences of splitting a column of A for the sparsity pattern of AA^T. As before, for ease of presentation we assume that column a_1 in (6) is completely dense and that it is cut into two equal parts

$$A = \begin{array}{|c|c|c|} \hline a_{11} & 0 & A_1 \\ \hline 0 & a_{12} & A_2 \\ \hline 1 & -1 & 0 \\ \hline \end{array} \quad . \tag{12}$$

If column a_1 of (6) is not completely dense, then the analysis may always be restricted to a submatrix of A built of those rows only which contain nonzero entries of a_1. From (12) we obtain

$$AA^T = \begin{array}{|c|c|c|} \hline \text{dense} & A_1A_2^T & a_{11} \\ \hline A_2A_1^T & \text{dense} & a_{12} \\ \hline a_{11}^T & a_{12}^T & 2 \\ \hline \end{array} \tag{13}$$

and the problem of finding optimal splitting becomes equivalent to choosing such a partition of a_1 that ensures minimum number of nonzero entries in parts $A_1A_2^T$ and $A_2A_1^T$ of (13). We may add here that when a_1 of (6) is to be cut into $p > 2$ equal parts, then it is done in $p-1$ successive steps each determining the partition like (12) into a_{11} containing k/p entries and a_{12} containing the rest of nonzeros such that the number of elements in $A_1A_2^T$ and $A_2A_1^T$ is minimized.

Let us observe that if A_0 of (6) is a staircase matrix, as for example in case of multistage stochastic or multicommodity network problems analysed by Lustig et al. (1989) and by Mulvey and Ruszczynski (1990), then its structure automatically determines optimal column partition for which $A_1A_2^T$

$= 0$ and $A_2 A_1^T = 0$.

Let us now pass to the presentation of the heuristic that determines optimal splitting. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ define disjoint subsets of row indices of these nonzero elements of column a_1 that have been included to a_{11} and a_{12}, respectively. The partition (12) is obtained in the iterative process which starts with a trivial splitting: $\mathcal{A}_1$ is empty and $\mathcal{A}_2$ is the list of row numbers of all nonzero entries of a_1. At every step of it one element is removed from list $\mathcal{A}_2$ and added to $\mathcal{A}_1$. The process is continued until $\mathcal{A}_1$ (and consequently a_{11}) has the required number of entries (k/2 if the column is to be split into two equal parts or k/p if it is to be cut into p equal parts). After $l-1$ steps (where $l-1 < k/2$) we may determine:

r_j the number of entries of j-th column of A_1 and

s_j the number of entries of j-th column of A_2, $j = 1,2,\ldots,n-1$

and the number of nonzeros in part $A_1 A_2^T$ of (13)

$$P = \sum_{j=1}^{n-1} r_j s_j \ . \tag{14}$$

In the next step an element of $\mathcal{A}_2$ is looked for such that moving it from $\mathcal{A}_2$ to $\mathcal{A}_1$ gives the largest possible reduction of the penalty term (14). For a nonzero entry $(a_{12})_i$ that appears in row i of a_{12} we thus determine

$$\delta P(i) = \sum_{j: w_{ij} \neq 0} (s_j - r_j - 1) \ , \tag{15}$$

where w_{ij} defines sparsity pattern of row i of A_2 i.e.:

$$w_{ij} = \begin{cases} 0 \ , & \text{if } a_{ij} = 0, \\ 1 \ , & \text{otherwise.} \end{cases} \tag{16}$$

It is easy to observe that $P + \delta P(i)$ gives the new value of penalty indicator (14) which will be obtained if an element i leaves $\mathcal{A}_2$ and is added to $\mathcal{A}_1$.

We then calculate $\delta P(i)$ for all elements that are still in a_{12}, determine i_* for which $\delta P(i)$ is minimum and "move" index i_* from $\mathcal{A}_2$ to $\mathcal{A}_1$.

Updating r_j and s_j completes iteration 1. This gives the following

Algorithm 1.

$\mathcal{A}_1 := \emptyset$

$\mathcal{A}_2 := \{i: (a_1)_i \neq 0\}$

$r_j := 0 , \quad j = 1,2,\ldots,n-1$

$s_j :=$ number of entries of j-th column of A_2 , $\quad j = 1,2,\ldots,n-1$

for $l = 1,2,\ldots,k/2$

for $i \in \mathcal{A}_2$

$$d_i := \sum_{j:w_{ij}\neq 0} (s_j - r_j - 1)$$

$$d_{i_*} := \min_{i \in \mathcal{A}_2} \{d_i\}$$

$\mathcal{A}_1 := \mathcal{A}_1 \cup \{i_*\}$

$\mathcal{A}_2 := \mathcal{A}_2 - \{i_*\}$

for $j = 1,2,\ldots,n-1$

if $w_{i_*j} = 1$ then

$r_j := r_j + 1$

$s_j := s_j - 1$

end if

The algorithm ends up with determining the partition of a_1 into a_{11} and a_{12}. The row indices defining this partition are in $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Let us observe that a comfortable access to the sparsity pattern of rows of A is necessary to implement the above algorithm efficiently. It is none restriction however since the access to the constraint matrix of the linear program both by columns and by rows is an elementary requirement satisfied by all codes based on interior point methods.

In our preliminary implementation the analysis is performed while reading MPS formatted data of the problem. All columns of A that are longer than a given threshold length are marked as those to be split. Additionally, space is left in internal data structures of the LP code for new short columns (parts of a long one) and for nonzero entries that will appear in the linking rows of type (3) bordered to a constraint matrix. Row linked lists of positions of row entries are created for the submatrix of A that contains only short columns. They allow a comfortable access to sparsity pattern of rows of this part of A that will remain unchanged as e.g. A_0 in (6) and ensure the efficiency of the execution of the two inner loops of Algorithm 1 (calculating d_i and updating r_j and s_j). When the reading of the constraint matrix is completed, all the earlier marked long columns are successively split and its short parts are added to internal data structures. The row linked lists are also updated after every short column addition so all the short columns (including the parts of already partitioned long ones) are taken into account when the optimal splitting of the next long column is looked for.

5. Numerical results

The method described has been tested on two real-life problems from Gay's (1985) *Netlib* collection (ISRAEL and SEBA) which are widely reported as particularly difficult for the reason of containing dense columns. Problem statistics are given in Table 1.

Table 1. Test problems.

Problem	ROWS	COLS	ELTS	average col. len.	maximum col. len.
ISRAEL	174	142	2269	16	136
SEBA	515	1028	4352	4	230

Table 2 presents the efficiency of the method of splitting dense columns applied to ISRAEL. Its first column contains the threshold length of the column of A such that any column longer than it has been split. The following columns contain: the number of added constraints, the number of nonzero elements of the lower triangular portion of the AA^T matrix after the splitting, the number of the subdiagonal nonzero elements of its Cholesky factor and the fill-in obtained during the factorization. Before the symbolic factorization was performed the matrix AA^T has been reordered by a symmetric permutation resulting from the minimum degree heuristic (see e.g., chapter 5 of the book of George and Liu (1981) or section 10.9 of the book of Duff et al. (1989)) that is supposed to minimize the fill-in reasonably. The last three columns of Table 2 present relative savings of storage for Cholesky factor, time of the minimum degree reordering and the symbolic factorization and time of the numerical factorization, respectively (direct method with no splitting determines a 100%). Table 3 collects analogous data for SEBA.

Table 2. Efficiency of splitting for ISRAEL.

THRESHOLD LENGTH	CONSTRAINTS ADDED	NONZEROS			RELATIVE SAVINGS (%)		
					STORAGE	TIME	
		AA^T	L	Fill-in		sym.fct	num.fct
∞	0	11053	11314	261	0	0	0
110	1	9838	10272	434	9	-8	18
100	2	9011	9643	632	15	-8	29
70	3	7263	8824	1561	22	-10	44
60	6	6371	8502	2131	25	-10	49
50	8	5819	7585	1766	33	+23	60
40	11	5544	8073	2529	28	+25	54
30	31	4879	8305	3426	27	+23	56
20	56	4772	8606	3834	24	+15	55

Table 3. Efficiency of splitting for SEBA.

THRESHOLD LENGTH	CONSTRAINTS ADDED	NONZEROS			RELATIVE SAVINGS (%)		
					STORAGE	TIME	
		AA^T	L	Fill-in		sym.fct	num.fct
∞	0	51400	53728	2328	0	0	0
220	2	50928	52987	2059	1	0	3
210	9	41389	44691	3302	17	14	9
200	9	41389	44691	3302	17	14	10
190	13	29483	33006	3523	39	34	30
180	14	28882	32192	3310	40	45	36
110	16	28587	31822	3235	41	45	43
100	23	24315	28437	4122	49	52	64
90	28	21154	24961	3807	54	63	76
70	37	18182	22994	4812	57	66	84
60	42	17299	21109	3810	61	69	88
50	51	15409	19489	4080	64	73	91
40	65	13656	18804	5148	65	75	92
30	93	12026	19293	7267	64	78	92
20	137	10299	27537	17238	49	72	82

It easily follows from the analysis of Tables 2 and 3 that the method presented gives significant memory savings when compared with direct solution of equation (2) by applying Cholesky factorization to the matrix AA^T constructed from the original constraint matrix of the problem. For example when all columns longer than 50 are split in the problem ISRAEL, then the Cholesky factor of AA^T has 33% less nonzero entries than the one of AA^T. Similarly, splitting all columns longer than 50 in SEBA reduces three times the number of nonzero elements of the Cholesky matrix. Time savings obtained are also significant since the reduction of nonzeros in AA^T remarkably simplifies the reordering, the symbolic factorization and, finally, significantly accelerates the numerical factorization. The application of the splitting technique reduced the time of the numerical phase of the decomposition by factors 2.5 and 12 for problems ISRAEL and SEBA, respectively.

Let us also mention that splitting alone took always about 5-10% of the time required by the reordering and the symbolic factorization so its contribution to the time of the whole solution process is negligible.

6. Conclusions

The method of splitting dense columns has the following advantages:

- it is easily implementable (preprocessing LP data);
- it never cause singularity of AA^T (assuming that the original problem is well formulated i.e. it does not have empty rows or linearly dependent constraints);
- it does not require any additional memory;
- it is fast.

It thus seems to be a useful option for including into any LP code based on logarithmic barrier approach.

References

Adler I., Karmarkar N., Resende M.G.C., Veiga G. (1989a) Data structures and programming techniques for the implementation of Karmarkar's algorithm, *ORSA Journal on Computing* **1**, No 2, pp. 84-106.

Adler I., Karmarkar N., Resende M.G.C., Veiga G. (1989b). An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming* **44**, pp. 297-335.

Choi I.C., Monma C.L., Shanno D.F. (1990). Further development of a primal-dual interior point method, *ORSA Journal on Computing* **2**, 304-311.

Duff I.S. Erisman A.M., Reid J.K. (1989) Direct methods for sparse matrices, Oxford University Press, New York 1989.

Gay D.M. (1985). Electronic mail distribution of linear programming test problems, Mathematical Programming Society COAL Newsletter.

George A., Liu J.W.H. (1981). Computer Solution of Large Sparse Positive Definite Systems, Prentice Hall, Inc., Englewood Cliffs, 1981.

Gill P.E., Murray W., Saunders M.A., Tomlin J.A., Wright M.H. (1986). On projected Newton barrier methods for linear programming and an equiva-

lence to Karmarkar's projective method, *Mathematical Programming* 36, pp. 183-209.

Gill P.E., Murray W., Saunders M.A. (1988). A single-phase dual barrier method for linear programming, Report SOL 88-10, Systems Optimization Laboratory, Stanford University, Stanford, 1988.

Golub G.H., Van Loan C.F. (1983). Matrix Computations, John Hopkins University Press, Baltimore 1983.

Hager W.W. (1989). Updating the inverse of a matrix, *SIAM Review* 31, No 2, pp. 221-239.

Lustig I.J., Mulvey J.M., Carpenter T.J. (1989). Formulation of stochastic programs for interior point methods, Technical Report SOR 89-16, Department of Civil Engineering and Operations Research, Princeton University, Princeton, October 1989.

Lustig I.J., Marsten R.E., Shanno D.F. (1990). On implementing Mehrotra's predictor-corrector interior point method for linear programming, Technical Report SOR 90-03, Department of Civil Engineering and Operations Research, Princeton University, Princeton, April 1990.

Mulvey J.M., Ruszczynski A. (1990). A diagonal quadratic approximation method for large scale linear programs, Technical Report SOR 90-08, Department of Civil Engineering and Operations Research, Princeton University, Princeton, September 1990.

Munksgaard N. (1980). Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients, *ACM Transactions on Mathematical Software* 6, No 2, pp 206-219.

Rockafellar R.T., Wets R.J.-B. (1987). Scenarios and policy aggregation in optimization under uncertainty, Working Paper WP-87-119, IIASA, Laxenburg.

Vanderbei R.J. (1990). Splitting dense columns in sparse linear systems, manuscript, September 1990.