INFORMATICA 4/91

DESIGN, IMPLEMENTATION AND TESTING OF A PRIMAL-DUAL INTERIOR POINT METHOD FOR SOLVING LINEAR PROGRAMS

Janez Barle and Janez Grad
Univerza v Ljubljani
Ekonomska fakulteta
Kardeljeva ploščad 17
61109 Ljubljana

Keywords: linear programing, primal-dual interior point algorithm, sparse Cholesky factorization

ABSTRACT: This paper describes our implementation of a primal-dual interior point algorithm for linear programming. The topics discussed include economic data structures, efficient methods for some sparse matrix operations, sparse Cholesky factorization, methods for handling dense columns and comparisons with simplex based methods. Extensive numerical results demonstrate the efficiency of the resulting algorithm as well as some problems which remain to be solved. The role of interior point based solvers in the process of solving large-scale mathematical programming models is also discussed.

SNOVANJE, IMPLEMENTACIJA IN TESTIRANJE PRIMARNO-DUALNE METODE NOTRANJE TOČKE ZA REŠEVANJE LINEARNIH PROGRAMOV: V sestavku je opisana naša implementacija primalno-dualne metode notranje točke za reševanje linearnih programov. Obravnavane so ekonomične podatkovne strukture, učinkoviti načini za izvajanje nekaterih operacij z razpršenimi matrikami, razpršeni razcep po Choleskem, metode za delo z gostimi stolpci ter primerjave z metodo simpleksov. Izčrpni rezultati numeričnega testiranja kažejo tako učinkovitost razvitih algoritmov, kot tudi nekatere probleme, ki jih je treba še razrešiti. Opisana je tudi vloga reševanja z metodami notranje točke v splošnem kontekstu modeliranja in reševanja velikih problemov matematičnega programiranja.

Introduction

The linear programming (LP) problem may be stated in the form:

$$\text{minimize (maximize) } c^T x$$
$$\text{subject to: } Ax = b, \quad l \le x \le u \qquad (1)$$

where A is a rectangular matrix, c, x, l, u, b are column vectors and the symbol ^{T} denotes transpose of a vector or matrix. Some of bounds in l and u may be infinite. The important features of typical LP constraint matrix A are its quite large dimension, sparsity and a specific structure i.e. patterns in which its nonzero elements appear. A classical way for solving the above problem is the simplex algorithm, developed by G.B. Dantzig in late 1940's. Contemporary variants of this algorithm, which are included in many commercial or university developed software packages, are tailored for solving quite large problems in a fast and reliable way. However, this established algorithm has got recently a serious competitor in the so called interior point methods. These methods became widely known after Karmarkar's publication (Karmarkar, 1984) of an algorithm that is claimed to be much faster for practical problems than the simplex method. Although these initial promises appeared to be too optimistic, Karmarkar's algorithm and other interior point methods are now regarded as a competitive methods for solving LP problems. This is particularly true when solving of some specific forms of super size problems on supercomputers is considered. Such kind of problems, which are often encountered in communication, transportation and military operations, are very sparse and usually exhibit specific and generally well-behaved block structures that can be effectively exploited. Efforts to develop software systems for solving super size LP problems with Karmarkar's algorithm proved to be very fruitful. One of the outstanding steps in this direction is AT&T's KORBX system. The system consists of both hardware, which uses parallel processing, and software which exploits the resources of this hardware (Carolan et al., 1990). However, there is also a need for exploring ability of interior point methods for solving LP on a more widely available serial computers. In the paper we present our work in this direction, which was performed on MS-DOS personal computers and VAX/VMS minicomputers.

Algorithms

Nowadays a plethora of research papers is published where different interior point methods are proposed. We have employed the variant of a primal-dual interior point method which is supposed to be among the most efficient (Lustig et al. 1989). In order to make clear differences between such kind of methods and simplex based algorithms, we first give a brief explanation of the revised simplex algorithm. The steps of this algorithm are roughly described within the following box where B denotes the basis matrix and c_B the cost vector of the basic variables. It is therefore assumed that there is a set of m basic variables, which is usually changed after each iteration in such a way that one nonbasic variable enters the basis and one basic variable leaves the basis. The informal description which follows is related to the second phase of the primal revised simplex algorithm, where the basic feasible solution is already known.

Revised-simplex-method

R1: Produce a pricing vector: $\pi = c_B^T B^{-1}$ (BTRAN).

R2: Select the entering variable x_s (column $u = A_{*s}$) according to a given pricing strategy. If no entering variable is found, terminate (solution is optimal).

R3: Update the entering column: $v = B^{-1}u$ (FTRAN).

R4: Determine the leaving variable. If none is found, terminate (problem is unbounded).

R5: Update the basis matrix representation; refactorize if necessary. Go to R1.

It must be noted however that there is not yet general agreement about what are the best algorithms in detail, and how they should be implemented in the most efficient way. In general, number crunching operations are concentrated within the steps R1 and R3 where two systems of linear equations have to be solved (these operations are often refered to as BTRAN and FTRAN). It is very important that after basis change updating of basis matrix is possible without performing full factorization, which has to be done only periodically. Other steps, particularly R2 and R4, deal mainly with logic decision and "book-keeping" problems. It is also obvious that a rather sophisticated data structures must be employed in order to exploit sparsity (Duff et al., 1989).

Interior point methods differ considerably from the simplex method. Primal-dual interior point method, which we have decided to implement, requires LP problem being formulated in the following form:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to:} && Ax = b,\quad x + s = u,\quad x \geq 0,\ s \geq 0 \end{aligned} \tag{2}$$

with the associated dual

$$\begin{aligned} &\text{maximize} && b^T y - u^T w \\ &\text{subject to:} && A^T y - w + z = c,\quad w \geq 0,\ z \geq 0 \end{aligned} \tag{3}$$

Fortunately, formulation (2) can be derived in a straightforward way from formulation (1). An outline of the algorithm is sketched within the following box, where X, S, W and Z are diagonal matrices with diagonal elements equal to the components of corresponding vectors x, s, w and z. Φ is the user supplied constant which is usually computed by using the following formula:

$$\Phi = \Phi(n) = \begin{cases} n^2 & n \leq 5000 \\ n\sqrt{n} & n > 5000 \end{cases}$$

and $M = \tau\Phi(n) * \max\{\|c\|_\infty, \|b\|_\infty\}$ where τ is a scalar multiplier which is used to allow variations of the initial μ. Furthermore, $d_D = A^T y^0 + z^0 - c$, where y^0 and z^0 are initial y and z, and $e = (1,1,\ldots,1)$. α_d, α_p are some appropriate step lengths in the primal and dual spaces respectively. These step lengths must be chosen in a way which ensures nonnegativity of variables x, s, z and w, for example:

$$\alpha_p = \alpha_0 * \min\{\min_j\{x_j/-\delta x_j,\ \delta x_j<0\},\ \min_j\{s_j/\delta x_j,\ \delta x_j>0\}\}$$

$$\alpha_d = \alpha_0 * \min\{\min_j\{z_j/-\delta z_j,\ \delta z_j<0\},\ \min_j\{w_j/-\delta w_j,\ \delta w_j<0\}\}$$

where $0<\alpha_0<1$ is the user supplied parameter which is usually set to be equal 0.9995.

Initial feasibility of the problem is formally assured by adding column $Ax_0 - b$ and row d_D to the matrix A, together with x_a and $y_a (=-z_b)$, which are corresponding primal and dual variable with initial value 1. In order to achieve feasibility their values must decrease to 0.

Primal-dual-interior-point-method

K1: Compute $\mu = (c^T x + u^T w - b^T y + M(x_a - y_a))/\Phi$

K2: Compute $\theta = (S^{-1}W + X^{-1}Z)^{-1}$

K3: Compute positive definite matrix $A\theta A^T$.

K4: Perform Cholesky factorization of the $A\theta A^T$

K5: Compute $\sigma(\mu) = \mu(S^{-1} - X^{-1})e - (W - Z)e$

K6: Compute
$$\begin{aligned} \delta y &= -(A\theta A^T)^{-1}(A\theta(\sigma(\mu) + z_b d_b) + (Ax - b)) \\ \delta x &= \theta(A^T\delta y + \sigma(\mu) - z_b d_b) \\ \delta s &= -\delta x \\ \delta z &= -\mu X^{-1}e + Ze - X^{-1}Z\delta x \\ \delta w &= -\mu S^{-1}e + We - S^{-1}W\delta x \end{aligned}$$

K7: Update
$$\begin{aligned} y_{new} &= y_{old} - \alpha_d \delta y \\ x_{new} &= x_{old} - \alpha_p \delta x \\ s_{new} &= s_{old} - \alpha_p \delta s \\ z_{new} &= z_{old} - \alpha_d \delta z \\ w_{new} &= w_{old} - \alpha_d \delta w \end{aligned}$$

K8: If relative duality gap satisfies relation:

$$\frac{c^T x + u^T w - b^T y}{1 + |u^T w - b^T y|} < \epsilon$$

where ϵ is user supplied constant, terminate (solution is optimal). Otherwise go to K1.

It was also assumed that the initial interior solution is supplied by the user. For example, it is possible to choose $x^0 = z^0 = \min\{e, u/2\}$ and $y_0 = 0$ (Choi et al., 1990). In general, interior point methods are not very sensible to the choice of the initial solution.

The above description is based on two papers (Lustig et al. 1989, Choi et al. 1990) where algorithmic aspects of the modularized fortran code OB1 (Optimization with Barriers 1) were described. Our intention was to develop our own code based on mentioned papers and standard methods for computing sparse Cholesky factorization (George, Liu, 1981). However, some of the implementation details are different, for example:

a) In our implementation column $Ax_0 - b$ and row d_D are computed at each iteration, rather than only for the initial solution. x_a and y_a are defined as ratios between current and initial ∞ norms of $Ax_0 - b$ and d_D. Furthermore, x_a and y_a retain some small value even in the case if their computed vsalue is zero. Such approach enable us to save some storage space and also, according to our experience, to improve accuracy of the computed solution.

b) Sometimes it is impossible for relative duality gap to reach prescribed ϵ on step K8. In such cases we terminate algorithm when relative difference between two subsequent objective values become smaller than the prescribed constant, which is usuallly set to be $0.1*\epsilon$.

On the whole, published description of the algorithm is good enough to enable everyone to create, possibly after some experimental investigation, workable implementation of a primal dual interior point method. Evidently the algorithm consists mostly of floating point computations and consequently fortran is an obvious choice of implementation language. Some features of the interior point methods that make them so much different from the simplex method are obvious:

1) There is no partitioning into basic and nonbasic variables. This means that, in principle, all variables and constraints are handled in equal way during the solution process.

2) Each iteration requires computationally expensive factorisation of positive definite matrix or solution of the least squares problem.

3) Solution vector x is always an interior point of the solution polytope.

Feature 1) has a far-reaching consequences. It can be a means for avoiding potential combinatorial problems arising in the movement from one basis to another which is typical for simplex method. On the other hand such approach may degrade computational speed and stability.

The main computational problem of the interior point methods is inversion of matrix $A\Theta A^T$ or solution of the corresponding linear least squares problem. This is usually done by computing sparse Cholesky factorization of $A\Theta A^T$. In order to understand methods for doing this, one must be acquainted with the methods for storing sparse matrices. In the next section the methods for storing sparse matrices which were applied in our implementation of primal-dual interior point methods are briefly described.

Data structures and implementation issues

Exploitation of sparsity is based on the fact that only nonzero elements of sparse matrix (or vector) must be stored, together with information about their position within matrix (vector). In the case of LP input data (A, b, c, u) within the framework of interior point methods, we have employed the following data structures:

1) Righthand-side vector b is stored as a dense vector.

2) Constraint matrix A is stored using three one dimensional vectors (XA,IA,CP) where

   XA = vector of nonzero values $A_{i,j}$ which are sorted by columns and (secondary) by row indices within a particular column, both in increased order.
   IA = vector of row indices of nonzero elements, which are sorted in a same manner as XA.
   CP = vector of column pointers which consists of locations where the representation of columns begins in XA and IA. For example, elements of column i are all in locations from CP(I) to CP(I+1)-1.

3) Nonzero elements of c are stored (formally) as n+1. column of A. Therefore they are stored between locations CP(N+1) and CP(N+2)-1 in XA (values) and IA (indices).

4) Noninfinite elements of u are stored (formally) as n+2. column of A. Therefore they are stored between locations CP(N+2) and CP(N+3)-1 in XA (values) and IA (indices).

Obviously, it is also necessary to store the matrix $A\Theta A^T$ and its triangular factor L, in the case if Cholesky factorization is used within the solution process. These matrices can be stored using the same storage locations. The amount of this storage is determined by fill-in which generally can not be avoided during the Cholesky factorization of $A\Theta A^T$. It is therefore advisable to try to minimize fill-in by appropriate reordering of rows and columns of $A\Theta A^T$. Ordering algorithms are essentially graph techniques for obtaining appropriate numbering of the graph nodes. In our case nonzero structure of $A\Theta A^T$ represents an undirected graph G(X,E) with m nodes. The adjacency list of node $x \in X$ is a list containing all nodes adjacent to x, which is represented by indices of nondiagonal nonzero elements of corresponding column of $A\Theta A^T$. The implementation of described structure is done by storing the adjacency lists sequentially in integer array ADJNCY along with an index vector XADJ of length M+1 containing pointers to the beginning of the lists in ADJNCY. The extra entry XADJ(M+1) points to the next available location in ADJNCY (George, Liu, 1981). These arrays are input data for ordering algorithms which can be generally divided in two groups:

a) reorderings which try to minimize number of nonzero elements (and therefore fill-in) in triangular factor L. Although it is known to be NP-complete problem (Duff et al. 1989) several reasonable good heuristics exist. One of them is the minimum degree algorithm. The name of this algorithm is derived from its graph theoretic interpretation: in the graph associated with a symmetric sparse matrix, this strategy corresponds to choosing that node for the next elimination which has the least edges connected to it.

b) reorderings which try to permute $A\Theta A^T$ and triangular factor L into some particular desirable form. This can be for example so-called envelope or profile form. The most known algorithm of this type is the Reverse Cuthill-McKee algorithm. The objective of such kind of algorithms is to reorder the rows and columns of the matrix so that the nonzeroes in the obtained matrix are clustered near the main diagonal since this property is retained in the corresponding Cholesky factor L. Such a cluster is called the profile or envelope and is defined to contain also all zero elements between the diagonal and the last nonzero element in the row or column. The problem of minimizing the envelope size of a matrix is proven to be NP-complete (Billionet, Breteau, 1989) and consequently the Reverse Cuthill-McKee algorithm is only one among heuristic procedures for doing this.

We have implemented both the minimum degree and the Reverse Cuthill-McKee algorithm within our LP package. In order to give some insight into these methods, we shall show how they perform on the smallest example from the NETLIB library (Gay, 1985), which is known under the name AFIRO. This is the problem with constraint matrix having 27 rows and 51 columns which contain all together 102 nonzero elements. Its corresponding $A\Theta A^T$ matrix has the structure as in the following picture, where only the upper triangular part is reproduced:

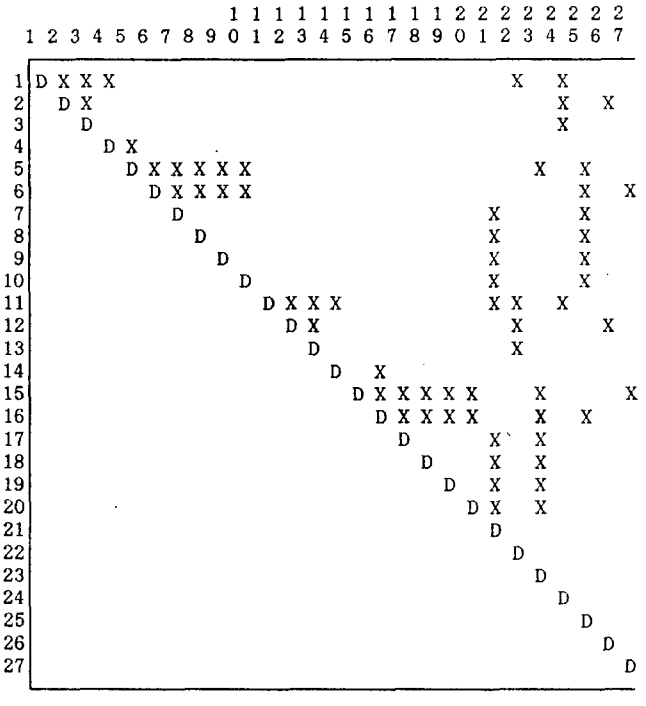

Figure 1. AFIRO - structure of the upper part of $A\Theta A^T$

Nondiagonal and diagonal nonzero elements are presented using symbols X and D respectively. It is obvious that, at least in this case, matrix $A\Theta A^T$ is not as sparse as matrix A itself. Moreover, number of its nonzeroes may substantially increase during the subsequent Cholesky factorization. The following picture displays how this situtation is controlled by applying the minimum degree algorithm. The produced ordering (permutation of rows and columns) is: (4,14,26,27,13,22,12,3,24,2,1,11,10,17, 18,19,20,23,16,15,9,8,25,21,6,7,5)

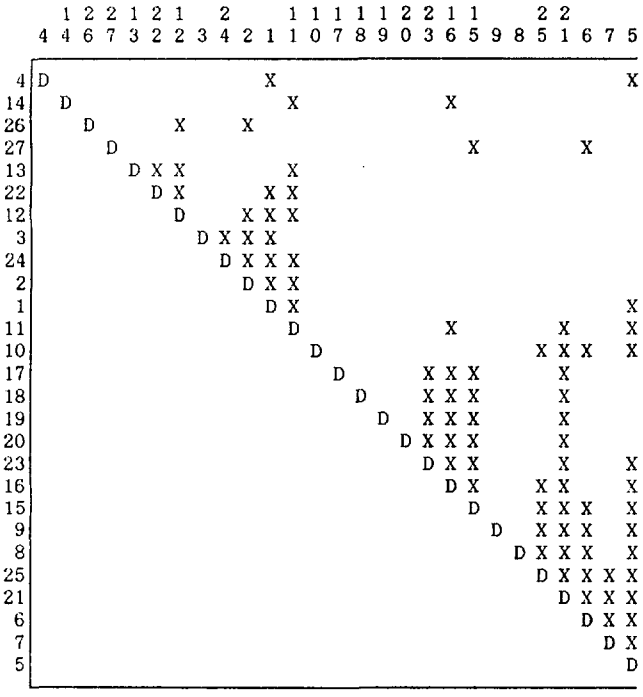

Figure 2. AFIRO - structure of the Cholesky factor

The above structure is determined not by actual numerical factorization but by simulation of it, or so called symbolic factorization. The advantage of this approach is that the data structures are set up once for all, as the structure of the matrix does not change from one iteration to another. Obviously, at each iteration it is necessary to perform numerical factorization since $A\Theta A^T$ is changing along with diagonal matrix Θ. The data structures returned by the symbolic factorization can be presented in a sparse storage scheme known as the compressed storage scheme of Sherman, cited in (George, Liu, 1981). The scheme has a storage array LNZ which will contain all nonzero entries in the nondiagonal part of Cholesky factor L column-wise (or, which are the same numbers, in the L^T row-wise), an INTEGER vector NZSUB which will hold the row subscripts of the nonzeroes, and an index vector XLNZ whose entries are pointers to the beginning of nonzeroes in each column in LNZ. The diagonal elements are stored separately in vector DIAG. In addition, an index vector XNZSUB is also used to hold pointers to the start of row subscripts in NZSUB for each column. This is the consequence of the key idea of the Sherman's compressed scheme: some of indices from NZSUB can be used for presenting nonzero patterns of two or even more columns. For example, it is applicable when columns 17,18,19,20 from the Figure 2 are considered.

Described data structure is filled et each iteration after computation of $A\Theta A^T$ is performed. The subsequent Cholesky factorization is performed using the same data structure. An interesting question is what is the best way to compute $A\Theta A^T$. At first sight row-oriented data structure for matrix A seems to be more practical and efficient when computation of $A\Theta A^T$ is considered. However, theoretical arguments (Duff et al. 1989) and practical testing convinced us that column-wise storage of A is much more efficient. The point is in avoiding operations with zero components of A. Perhaps this can be best explained with our fortran code for computating $A\Theta A^T$, which follows.

```
C*************************************************************
C     At this moment LNZ and working vector Z must be
C     initialised to zero values.
      DO 500 ICOL=1,N
         ISTART = CP(ICOL)
         ISTOP = CP(ICOL+1) - 1
         DO 100 I=ISTART,ISTOP
            J = IA(I)
            Z(J) = XA(I)*THETA(ICOL)
            DIAG(J) = DIAG(J) + Z(J)*XA(I)
  100    CONTINUE
         DO 200 I=ISTART,ISTOP-1
            IROW = IA(I)
            Z0 = Z(IROW)
            L = XLNZ(IROW)
            KSUB = XNZSUB(IROW)
            JBEGIN = I + 1
            DO 150 J=JBEGIN,ISTOP
  125          CONTINUE
               IF (NZSUB(KSUB) .EQ. IA(J)) THEN
                  LNZ(L) = LNZ(L) + Z0*XA(J)
               ELSE
                  L = L + 1
                  KSUB = KSUB + 1
                  GOTO 125
               ENDIF
  150       CONTINUE
  200    CONTINUE
         DO 300 I=ISTART,ISTOP
            Z(IA(I)) = 0.0
  300    CONTINUE
  500 CONTINUE
C*************************************************************
```

Another ordering algorithm which we have implemented is the Reverse Cuthill-McKee algorithm. Its performance on our test problem is presented on the following picture.

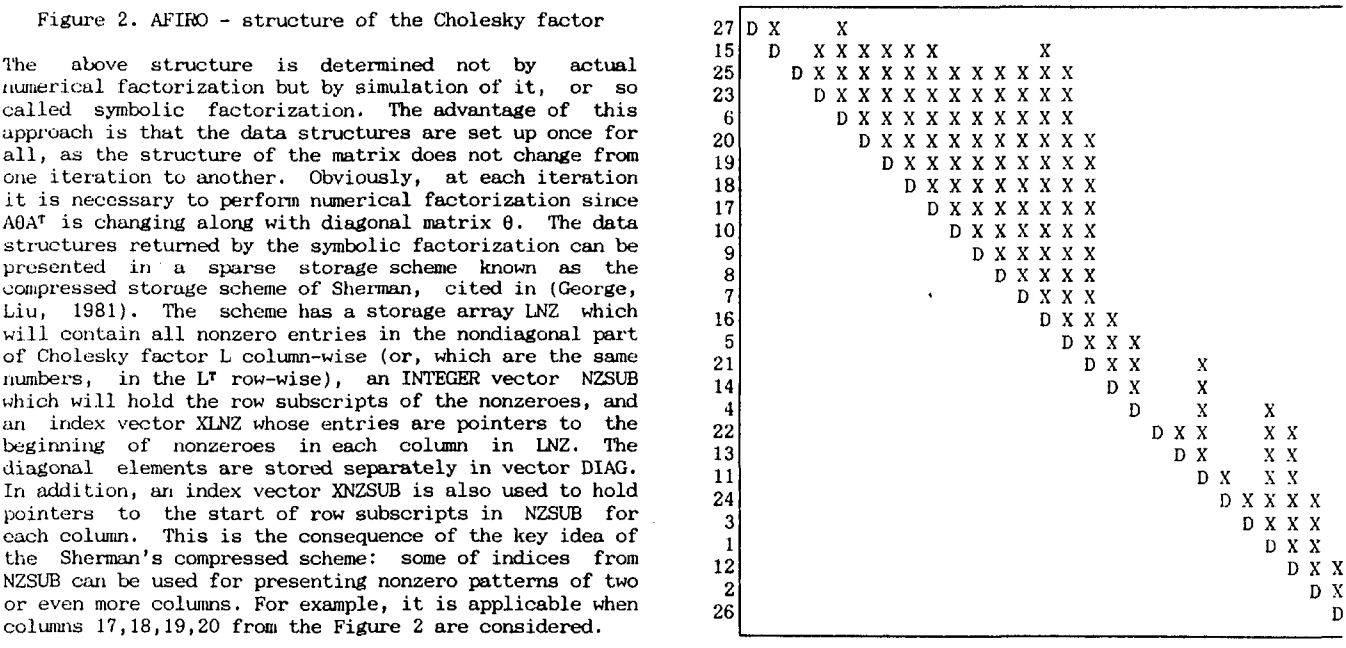

Figure 3. AFIRO - Envelope after RCM algorithm

The storage scheme for envelope methods has a main storage array ENV which will contain all entries (zeroes as well as nonzeroes) between the first nonzero entry in row of L (or column of L^T) and the diagonal, an index vector XENV whose entries are pointers to the beginning of nonzeroes in each row of L, and a vector DIAG where diagonal entries are stored.

Our practical experience with the Reverse Cuthill-McKee algorithm was quite disappointing. Its corresponding storage scheme is usually not as economical as those produced by the minimum degree algorithm. The results concerning speed of the numerical factorization were even less competitive. However, it would be interesting to try other profile methods. Some of them produce more economical storage scheme than the Reverse Cuthill-McKee algoritm (Billionet, Breteau, 1989). There are also some other efficient ordering algorithms, for example the nested dissection ordering algorithm (George, Liu, 1981). Their quantitative and qualitative comparisons may be an interesting topics for further research.

Handling of dense columns

The problem is what to do if one or more columns of A are dense vectors. In such cases computation of $A\Theta A^T$ leads to a very dense matrix and therefore should be avoided, if it is possible. This situation can be overcome by first dividing columns of A into dense and sparse submatrices:

$$A = [A_s ; A_d] \quad \text{and consequently} \quad \theta = [\theta_s ; \theta_d]$$

and then performing incomplete Cholesky factorization:

$$LL^T = A_s \theta_s$$

After this is done several approaches are possible. One among them is to use the incomplete Cholesky factors as preconditioner for a so called conjugate gradient algorithm (Adler et al. 1989). Another way is to use a method which deals with "dense windows" (Andersen et al. 1990). This method solves equation $A\Theta A^T y = q$ by performing the following operations:

Compute $V = A_d \theta_d$

Set up

$$\begin{bmatrix} LL^T & -V \\ V^T & I \end{bmatrix} \begin{bmatrix} y \\ \delta \end{bmatrix} = \begin{bmatrix} q \\ 0 \end{bmatrix}$$

Solve for δ by dense Cholesky factorization:

$$[I + V^T (LL^T)^{-1} V]\delta = -V^T (LL^T)^{-1} q$$

Compute $y = (LL^T)^{-1}(q+V\delta)$

It is obvious that typically there are only a few dense columns in A and therefore computing and storing dense Cholesky factorization is a trivial task.

Unfortunately, it was pointed out (Lustig et al., 1989) that removing dense columns can severely exacerbate the problem of ill-conditioning on badly conditioned problems. For this reason a search of reliable methods for handling dense columns remains an open research problem. One among possible approaches is to exploit fact that LP problem generally can be formulated in many different but equivalent ways. For example, it is possible to split dense column into two or more new columns which may result in sparser $A\Theta A^T$ (Gondzio, 1991). Just to give impression about that approach, we note that the following two matrices represent two different but equivalent formulations:

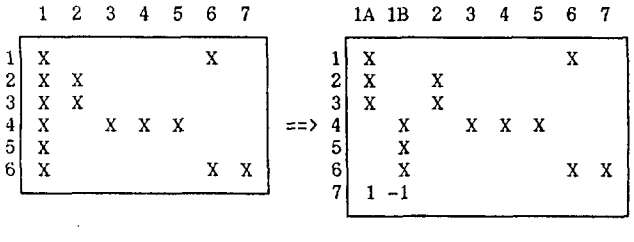

Figure 4. Column splitting

The first formulation will lead to completely dense $A\Theta A^T$, while this is not so for the second one.

Computational results

In this section, we report the computational results of running our implementation of a primal-dual interior point method on a set of LP test problems available through NETLIB (Gay, 1985). NETLIB is a system designed to provide efficient distribution of public domain software to the scientific community through different computer networks. We considered in our tests all 53 problems which are currently available to us. However, 3 of them (CZPROB, 80BAU3 and PILOTS) we were not able to read from the input file due to insufficient generality of our subroutine for reading LP data written in MPS format. Therefore we have used only 50 problems in our tests. Some quantitative data related to their size and storage consumption (when the minimum degree algorithm is used) are presented in the following table.

Problem:	No:	m:	n:	nz(A):	nz(AA^T):	nz(L):	nz(ind):
25FV47	45	820	1876	10705	11894	34590	8925
ADLITTLE	2	56	138	424	384	355	171
AFIRO	1	27	51	102	90	80	46
BANDM	19	305	472	2494	3724	4355	1913
BEACONFD	24	173	295	3408	2842	2727	1783
BORE3D	9	233	333	1446	2424	2860	1331
BRANDY	14	193	303	2202	2734	3236	1306
CAPRI	11	271	480	1933	3112	5569	2628
E226	20	223	472	2768	2823	3416	1390
ETAMACRO	16	400	734	2188	2771	11186	4016
FFFFF800	34	524	1028	6401	10615	18520	6601
GANGES	35	1309	1706	6937	8965	29359	7080
GFRDPNC	23	616	1160	2445	1451	1537	1270
GROW15	32	300	645	5620	3430	5790	5372
GROW22	38	440	946	8252	5040	8590	8039
GROW7	18	140	301	2612	1590	2590	2324
ISRAEL	15	174	316	2443	11227	11259	1373
NESM	47	662	2930	13260	4743	21283	8604
PILOT4	29	410	1181	7242	6743	14273	6781
PILOTJA	49	924	2044	13339	14174	50793	12560
PILOTWE	40	722	2930	9537	5547	15422	6109
RECIPE	6	87	178	652	582	584	153
SC205	3	205	317	665	656	986	701
SCAGR25	12	471	671	1725	2393	2510	1521
SCAGR7	4	129	185	465	629	636	417
SCFXM1	17	330	600	2732	3233	4400	1797
SCFXM2	30	660	1200	5469	6486	8977	3673
SCFXM3	36	990	1800	8206	9739	13631	5556
SCORPION	10	388	466	1534	2101	2086	937
SCRS8	26	490	1275	3288	2198	6117	2791
SCSD1	22	77	760	2388	1133	1315	464
SCSD6	31	147	1350	4316	2099	2398	960
SCSD8	46	397	2750	8584	4280	5482	1682
SCTAP1	13	300	660	1872	1686	2261	1430
SCTAP2	37	1090	2500	7334	6595	13729	6719
SCTAP3	43	1480	3340	9734	8866	17156	8636
SEBA	27	515	1036	4360	51915	53695	6176
SHARE1B	8	117	253	1179	1001	1345	526
SHARE2B	5	96	162	777	871	925	350
SHELL	28	536	1527	3058	1991	3556	2099
SHIP04L	39	360	2166	6380	4588	4428	1608
SHIP04S	33	360	1506	4400	3272	3252	1198
SHIP08L	50	712	4363	12882	9224	8948	3224
SHIP08S	42	712	2467	7194	5440	5464	2160
SHIP12L	51	1042	5533	16276	11715	11193	4742
SHIP12S	44	1042	2869	8284	6387	6289	2063
SIERRA	41	1222	2715	7951	6118	11665	5742
STAIR	25	356	538	3831	6653	15281	6263
STANDATA	21	359	1258	3173	1758	2726	1505
VTPBASE	7	198	329	945	1773	2217	974

Table 1. Quantitative data about LP problems

Numbers in the second column are related to the sequence of problems ordered by the number of nonzeroes. The last two columns show numbers of used entries in LNZ and NZSUB resceptively. On the one hand these data show that usage of compressed storage scheme is very efficient, but on the other hand indicate that density of triangular Cholesky factor can be a serious problem. In general, storage consumption of our implementation can be estimated using the following approximate formula:

real numbers (REAL*8)...: 6n + 4m + nz(A+c+u) + nz(L)
integers.....: n + 4m + nz(A+c+u) + nz(ind)

where nz(A+c+u) is the number of nontrivial elements in A,c and u together. It must be noted that interior point methods generally consume more storage than simplex based methods. Nevertheless, use of the above formula

shows that about 40 out of 50 problems can be solved on a standard PC with no more than 640 KB memory.
Our computational testing was performed on a VAX8550 computer. The operating system was VMS, version 5.2, and the VMS FORTRAN compiler, version 5.2, was used with the default options. The algorithm was used with default parameters α_0=0.9995, τ=0.1 and ϵ=10^{-8}. Computed optimal objective value and number of iterations were compared with those obtained using OB1 (Lustig et al. 1989).

Problem:	Iter. (Lustig)	Computed opt. value	Relative d. gap	Opt. value Lustig et al.
25FV47	48 (48)	5501.8459	0.321e-06	5501.8459
ADLITTLE	21 (17)	225494.96	0.447e-07	225494.96
AFIRO	14 (13)	-464.75314	0.249e-08	-464.75314
BANDM	31 (28)	-158.62802	0.579e-07	-158.62802
BEACONFD	25 (21)	33592.486	0.427e-07	33592.486
BORE3D	28 (25)	1373.0804	0.184e-08	1373.0804
BRANDY	34 (27)	1518.5100	0.412e-07	1518.5099
CAPRI	40 (37)	2690.0165	0.443e-05	2690.0127
E226	34 (31)	-18.751929	0.371e-07	-18.751929
ETAMACRO	51 (52)	-755.71523	0.132e-07	-755.71523
FFFFF800	66 (59)	555679.61	0.869e-06	555679.56
GANGES	41 (33)	-109585.74	0.670e-08	-109585.74
GFRDPNC	30 (26)	6902236.0	0.896e-07	6902236.0
GROW15	34 (25)	-1.0687094e+8	0.105e-07	-1.0687094e+8
GROW22	32 (30)	-1.6083431e+8	0.347e-06	-1.6083434e+8
GROW7	30 (22)	-47787812.	0.519e-08	-47787812.
ISRAEL	47 (47)	-896644.82	0.593e-07	-896644.82
NESM	70 (66)	14076037.	0.218e-04	14076036.
PILOT4	58 (56)	-2581.1378	0.779e-06	-2581.1405
PILOTJA	67 (67)	-6113.1349	0.155e-06	-6113.1365
PILOTWE	74 (71)	-2.7201067e+6	0.728e-06	-2.7201075e+6
RECIPE	18 (16)	-266.61600	0.867e-08	-266.61600
SC205	22 (16)	-52.202061	0.785e-07	-52.202061
SCAGR25	28 (24)	-14753433.	0.132e-06	-14753433.
SCAGR7	22 (22)	-2331389.8	0.316e-07	-2331389.8
SCFXM1	38 (31)	18416.759	0.192e-06	18416.759
SCFXM2	38 (37)	36660.263	0.149e-06	36660.262
SCFXM3	41 (39)	54901.255	0.194e-06	54901.255
SCORPION	21 (18)	1878.1250	0.141e-06	1878.1248
SCRS8	51 (50)	904.29696	0.453e-07	904.29695
SCSD1	14 (12)	8.6666667	0.320e-07	8.6666667
SCSD6	15 (15)	50.500000	0.638e-08	50.500000
SCSD8	14 (15)	905.00001	0.998e-08	905.00000
SCTAP1	22 (22)	1412.2500	0.643e-10	1412.2500
SCTAP2	23 (23)	1724.8071	0.809e-08	1724.8071
SCTAP3	27 (26)	1424.0000	0.150e-08	1424.0000
SEBA	32 (29)	15711.600	0.470e-07	15711.600
SHARE1B	42 (40)	-76589.318	0.254e-07	-76589.319
SHARE2B	20 (17)	-415.73224	0.778e-07	-415.73224
SHELL	39 (37)	1.2088253e+9	0.116e-06	1.2088253e+9
SHIP04L	26 (22)	1793324.5	0.161e-06	1793324.5
SHIP04S	22 (21)	1798714.7	0.124e-06	1798714.7
SHIP08L	26 (24)	1909055.2	0.111e-06	1909055.2
SHIP08S	25 (23)	1920098.2	0.856e-07	1920098.2
SHIP12L	29 (27)	1470187.9	0.139e-06	1470187.9
SHIP12S	29 (27)	1489236.1	0.161e-06	1489236.1
SIERRA	30 (26)	15394362.	0.140e-06	15394362.
STAIR	27 (25)	-251.26695	0.187e-06	-251.26695
STANDATA	34 (28)	1257.6995	0.295e-07	1257.6995
VTPBASE	28 (24)	129831.46	0.414e-08	129831.46

Table 2. Computational results

Our testing was therefore successful in a sense that we have solved all 50 problems with reasonable accuracy. However, we have experienced numerical troubles (such as floating overflow or underflow) on some problems:

1) To solve problem SCFXM1 we had to use α_0=0.95.
2) To solve problems PILOT4, PILOTWE and PILOTJA, which are known to be very ill-conditioned, we had to use α_0=0.95 again and also to change initial τ to value τ=0.001.

Although we have eventually solved above problems after some experimentation with algorithm parameters, a sound solution would be to use some kind of scaling. This means that instead of $A\Theta A^T$, in some cases it is better to work with the matrix $RA\Theta A^TR$, where R is a suitable chosen diagonal matrix.

On the whole, we were not able to obtain such quality of solution as it was reported for OB1, neither in terms of number of iterations, nor in terms of obtained relative duality gap. This is easily explainable by the fact that many fine details and important options, which are present in OB1, are not yet implemented in our code.

We have performed also some other types of computational testing, just to check validity of some assumptions about interior point methods. For example:
- A good simplex code usually outperforms interior point methods on small problems.
- Some problems, which are supposed to be difficult for simplex based methods, can be easily (and much faster) solved if some interior point method is used. For example, this is the case with problem 25FV47.
- Computational work within a step of interior point methods is dominated by sparse Cholesky factorization of $A\Theta A^T$. Its share on bigger problems is 60% to 90% of overall time.

System design and implementation

Implementations of interior point method were done in a form of highly portable fortran modules LPENV and LPMDG. The former uses the Reverse Cuthill-McKee algorithm, the latter uses the minimum degree algorithm. They are still in the phase of testing and development, for which we are using mainly VAX/VMS computer system, although the same code is running also on the PC. Our ultimate goal is to create reliable, portable and user-friendly LP software package based on the interior point algorithms, which is to be called LPINT. In our opinion such kind of system must contain portable full screen user interface and also subroutines for graphical display of different LP matrices (Alvarado, 1990). It is also very important to allow user to include his/her subroutines into the package. Data exchange between a user program and LPINT may be done with the usage of some type of communication region (CR). An overall LPINT system architecture can be ilustrated with the following picture:

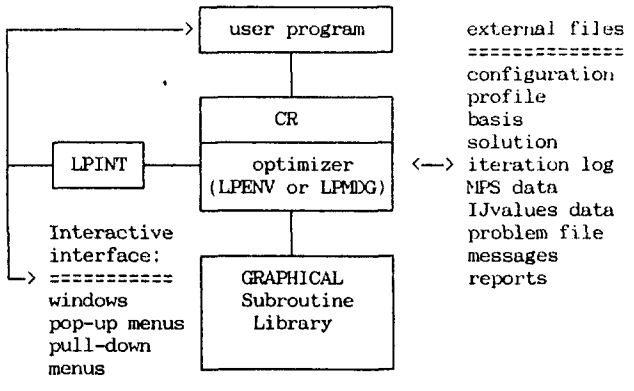

Figure 5. LPINT system architecture

Conclusions

Generally we can say that the interior point methods are getting more and more reliable and sophisticated as well. Moreover, interior point methods had greatly influenced algorithmic and experimental work in the field of linear programming. However, it is not likely that interior point methods can completely replace the simplex method in future. In our opinion reasons for this are mainly in simplex method ability to produce optimal basic solution. The knowledge of basic status of variables is very important, especially when postoptimal analysis or solution of mixed integer programs are considered. On the other hand, when one wish to solve big LP problem for the first time, it is advisable to start with some interior point based package. It is a

fast, reliable and robust way to obtain the first information about LP model.

A primal-dual interior point method can be implemented in a rather easy and straightforward way. This fact, as well as method's efficiency and mathematical elegance, can be a big step toward deeper understanding of interior point methods in general.

References

1. Adler, I., Karmarkar, N., Resende. M. and Veiga, G. (1989), "An Implementation of Karmarkar's Algorithm for Linear Programming", Mathematical Programming, Vol. 44, pp. 297-335.

2. Alvarado, F.L. (1990), "Manipulation and visualization of sparse matrices", ORSA Journal on Computing, Vol. 2, pp. 187-207.

3. Andersen, J., Levkovitz, R., Mitra, G. and Tamiz, M. (1990), "Adopting Interior Point Algorithm for the Solution of LPs on Serial, Coarse Grain Parallel Computers". Presented at the International Symposium on Interior Point Methods for Linear Programming: Theory and Practice, January 18-19, 1990, Europe Hotel, Scheveningen, Netherlands.

4. Billionnet, A. and Breteau, J.F. (1990), "A Comparison of Three Algorithms for Reducing the Profile of a Sparse Matrix", Recherche Opérationnelle, Vol. 23, pp. 289-302.

5. Carolan, W., Hill, J., Kennington, J., Niemi, S. and Witchman S. (1990), "An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications", Operations Research, Vol. 38, pp. 240-248.

6. Choi, C., Monma, C.L. and Shanno, D.F. (1990), "Further Development of a Primal-Dual Interior Point Method", ORSA Journal on Computing, Vol. 2, pp. 304-311.

7. Duff, I.S., Erisman, A.M. and Reid, J.K. (1989), Direct Methods for Sparse Matrices, Clarendon Press, Oxford.

8. Gay, D.M. (1985), "Electronic Mail Distribution of Linear Programming Test Problems", Mathematical Programming Society COAL Newsletter, Vol. 13, pp. 10-12.

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

10. Gondzio, J. (1991), "A Method for Handling Dense Columns of Linear Programs in the Interior Point Algorithm", Presented at the International Symposium "Applied Mathematical Programming and Modelling", London.

11. Karmarkar, N.K. (1984), "A New Polinomial-Time Algorithm for Linear Programming", Combinatorica, Vol. 4, pp. 373-395.

12. Lustig, I.J., Marsten, R.E. and Shanno, D.F. (1989), "Computational Experience with a Primal-Dual Interior Point Method For Linear Programming", Technical Report SOR 89-17, School of Industrial Engineering and Operations Research, Georgia Institut of Technology, Atlanta.