18^o Συνέδιο της Ε.Ε.Ε.Ε. – «Διαχείριση Κινδύνων» .

Some computational results on the efficiency of an exterior point algorithm

El-Said Badr[1], K. Paparrizos, Baloukas Thanasis, G. Varkas

University of Macedonia, Department of Applied Informatics, 156 Egnatia Str., 54006 Thessaloniki, E-mail: {it02185, thanasis, paparriz }@uom.gr

Abstract

The aim of this paper is to present some computational results for the exterior point simplex algorithm (EPSA). Our implementation was carried out under the C environment. This algorithm seems to be more efficient than the classical primal simplex algorithm (PSA), employing Dantzig's pivoting rule. Preliminary computational studies on randomly generated sparse linear programs support this belief. Also we use a modification of a recently developed procedure for updating inverse matrices and incorporate it with EPSA. This calculation requires $\Theta(m^2)$ operations .In our computational study we use the modification of the product form of the inverse which is faster than the classical product form of the inverse.

.

Keywords: Linear programming: Exterior point algorithm; Algorithm evaluation; Numerical experiments

[1] Research supported by the Greek Scholarship Foundation, (I.K.Y).

18^o Συνέδιο της Ε.Ε.Ε.Ε. – «Διαχείριση Κινδύνων» .

Μερικά υπολογιστικά αποτελέσματα για την αποδοτικότητα ενός εξωτερικού αλγορίθμου σημείου

El-Said Badr, K. Paparrizos, Baloukas Thanasis, G. Varkas

University of Macedonia, Department of Applied Informatics, 156 Egnatia Str., 54006

Thessaloniki, E-mail: {it02185, thanasis, paparriz }@uom.gr

Περίληψη

Ο στόχος αυτού του άρθρου είναι να παρουσιαστούν μερικά υπολογιστικά αποτελέσματα για τον εξωτερικό μονοκατευθυντικό αλγόριθμο σημείου (EPSA). Η εφαρμογή μας πραγματοποιήθηκε στο πλαίσιο του περιβάλλοντος C. Αυτός ο αλγόριθμος φαίνεται να είναι αποδοτικότερος από τον κλασσικό πρωταρχικής σπουδαιότητας μονοκατευθυντικό αλγόριθμο (PSA), υιοθέτηση Dantzig's να περιστρέψει κανόνας. Οι προκαταρκτικές υπολογιστικές μελέτες για τα τυχαία παραγμένα αραιά γραμμικά προγράμματα υποστηρίζουν αυτήν την πεποίθηση. Επίσης χρησιμοποιούμε μια τροποποίηση μιας πρόσφατα αναπτυγμένης διαδικασίας για τις αντίστροφες μήτρες και την ενσωματώνουμε με EPSA. Αυτός ο υπολογισμός απαιτεί $\Theta(m^2)$ διαδικασίες . Ι ν η υπολογιστική μελέτη μας χρησιμοποιούμε την τροποποίηση της μορφής προϊόντων του αντιστρόφου που είναι γρηγορότερο από την κλασσική μορφή προϊόντων του αντιστρόφου.

.

Λέξεις κλειδιά: Γραμμικός προγραμματισμός: Εξωτερικός αλγόριθμος σημείου; Αξιολόγηση αλγορίθμου; Αριθμητικά πειράματα

1. Introduction

The simplex method has been studied extensively since its invention in 1947 by G.B. Dantzig and still remains one of the most efficient methods for solving a great majority of practical problems. Although a number of variants of the simplex method have been developed, none of them has polynomial time complexity [16]. That is, the simplex method may require computational effort which grows exponentially with the size of the given problem.

The first polynomial algorithm was invented by Khachian in 1979 and is called the ellipsoid algorithm.The practical performance of this algorithm is poor for practical problems compared with the simplex method. In 1984, Karmarkar presented a new polynomial algorithm for the linear problems (LP) which approaches the optimal solution from the interior of the feasible region [7]. Since this Karmarkar's seminal paper was published, many papers have been written about interior point methods [14, 17]. As their theoretical properties show, interior point methods work well for real world problems in real life and outperform the simplex method for very large-scale problems [8].

EPSA was originally developed by Paparrizos [9] for assignment problem. Latter, Paparrizos [10] generalized his exterior point method to the general linear problem by developing a dual, in nature, algorithm. Independently, Anstreicher and Terlaky [1], developed a similar primal algorithm solving a sequence of linear sub-problems. Another relevant result was developed by Chen et al. [4].

A common feature of almost all-simplex type algorithms is that they can be interpreted as a procedure following simplex type paths that lead to the optimal solution. This algorithm differs radically from the Primal Simplex Algorithm (PSA) because its basic solutions are not feasible. It has been pointed out by Paparrizos et al. [12] that the geometry of EPSA reveals that this algorithm is faster than the well-known simplex method, a fact that was verified by preliminary computational results comparing early dual versions of EPSA on specially structured linear problems (see[6]). Recently, Paparrizos et al.[13] presented computational results relating that EPSA is more efficient than the classical primal simplex algorithm PSA.

In this paper, our aim is to study the standard linear programming form. In section 2, we recall some well-known facts about the simplex algorithm in revised form. As it is the first algorithm that solves linear programs it is very well studied in the literature. Also, we present the EPSA and a modification of a recently developed procedure for updating inverse matrices and incorporate it with exterior point algorithm. In Section 3 we give the computational results that demonstrate the superiority of our algorithm over simplex algorithm on a class of randomly generated problem. Finally, in Section 4 we give our conclusions and discuss possible extensions of the algorithm.

2. Algorithm description

We concerned with the following linear programming program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0, \end{array} \qquad \text{(P.1)}$$

where A$A \in \mathbb{R}^{m \times n}, c, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ and T denotes transposition. We find it convenient to describe the algorithm using the revised form of the simplex algorithm. Let us first assume that *A* is full rank, rank(*A*)=*m*, 1 $\leq m \leq n$. We will follow a time-honored traditional abuse of notation and write *B* instead of A_B and *N* instead of A_N. Whether *B* and N stands for a subset of indices or a matrix should be clear for the context. With the matrix *A* partitioned as *A*= (*B N*) and with a corresponding partitioning and ordering of *x* and *c*

$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$, $c = \begin{bmatrix} c_B \\ c_N \end{bmatrix}$ (P.1) is written as

$$\begin{array}{ll} \min & c_B^T x_B + c_N^T x_N \\ \text{subject to} & Bx_B + Nx_N = b \\ & x_B, x_N \geq 0 \end{array}$$

Here *B* is an $m \times m$ non-singular submatrix of *A* called basic matrix. The columns of *A* which belong to *B* are called basic and the remaining ones are called non-basic. Given a basic *B* the associated solution $x_B = B^{-1}b$, $x_N = 0$ is called a basic solution. A solution x=(x_B , x_N) is feasible if $x \geq 0$. Otherwise it is called infeasible. It is well known that the solution of the dual problem, which corresponds to the basis B, is given by $s = c - A^T w$ where $w^T = (c_B)^T B^{-1}$ are the simplex multipliers and s are the dual slack variables. The associated basis *B* is called dual feasible if $s \geq 0$. It is well known that $s_B = 0$.

In every iteration PSA interchanges a column of the current *B* with a column of the current non-basic *N*, constructing in this way a new basis. Geometrically, this means that PSA moves along the edges of the polyhedron $P = \{x \mid Ax \leq b, x \geq 0\}$. Such a path is known as a simplex path. On the other hand EPSA generates two paths toward the optimal solution. One path is infeasible (exterior) and the other is feasible. So EPSA does not need to proceed by following one such edge after another along the polyhedron. Therefore, avoiding the feasible region we can follow shorter paths to the optimal solution. Note that the total work of one iteration of simplex type algorithms is dominated by the determination of the inverse matrix B^{-1}. This inverse however, does not have to be computed from scratch in each iteration. The next inverse $(B^{-1})^{(r+1)}$ can be computed from the current inverse $(B^{-1})^{(r)}$ with a simple pivoting operation. Namely we have

$$(B^{-1})^{(r+1)} = E^{-1}(B^{-1})^{(r)} \tag{1}$$

Where E^{-1} is the inverse of an eta-matrix. We use relation (2) to compute E^{-1}.

$$E^{-1} = I - \frac{1}{a_{pq}}(a_q - e_q)e_q^T \tag{2}$$

In the above relation a_{pq} denotes the pivot element, column *q* is called the pivot column and row *p* is called the pivot row. A formal description of the primal EPSA in the revised form is given below. The above technique is called product form of the inverse.

2.1. **EPSA algorithm**

Step 0 (Initialization).

Start with a feasible basic partition (*B*,*N*).

Compute the vectors and matrices B^{-1}, x_B, w, s_N. Find the sets $P = \{j \in N : s_j < 0\}$ and $Q = \{j \in N \ s_j \geq 0\}$ choose an arbitrary vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_{|P|}) > 0$, compute s_0 using the relation $s_0 = \sum_{j \in P} \lambda_j s_j$ and the vector $d_B = -\sum_{j \in P} \lambda_j h_j$ with $h_j = B^{-1}A_{\cdot j}$

Step 1 (Test of termination)

(i) (Test of optimality) If $P = \varnothing$, STOP. Problem (P.1) is optimal.

(ii) (Choice of leaving variable). If $d_B \geq 0$, STOP. If $s_0 = 0$, problem (P.1) is optimal. Otherwise, choose the leaving variable $x_{B[r]} = x_k$ using the relation

$$\alpha = \frac{x_{B[r]}}{-d_{B[r]}} = \min \left\{ \frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0 \right\}$$

If $\alpha = +\infty$ problem (P.1) is unbounded. Put $y_B = x_B + \alpha\, d_B$

Step 2 (Choice of entering variable).

Compute the vectors $H_{rP} = (B^{-1})_{r.} A_{.P}$ and $H_{rQ} = (B^{-1})_{r.} A_{.Q}$. Also find the ratios θ_1 and θ_2 using the relations

$$\theta_1 = \frac{-s_P}{h_{rP}} = \min \left\{ \frac{-s_j}{h_{rj}} : h_{rj} > 0 \text{ and } j \in P \right\}$$

$$\theta_2 = \frac{-s_Q}{h_{rQ}} = \min \left\{ \frac{-s_j}{h_{rj}} : h_{rj} < 0 \text{ and } j \in Q \right\}$$

and determine indexes t_1 and t_2 such that $P(t_1) = p$ and $Q(t_2) = q$. If $\theta_1 \leq \theta_2$, set $l = p$. Otherwise, set $l = q$.The non-basic variable x_l enter in the basis.

Step 3 (Pivoting).

Set $B[r] = l$. If $\theta_1 \leq \theta_2$ set $P = P \setminus \{l\}$ and $Q = Q \cup \{k\}$. Otherwise, set $Q[t] = k$.

Using the new partition (B, N), where $N = (P, Q)$, update the vectors and matrices B^{-1}, x_B, w, s_N , $d_B = y_B - x_B$

. Go to step 1.

2.2 A Modification of Product Form of the Inverse

To justify the next algorithm and analysis we need the following notations:

$x \otimes y$	:	outer product of the vectors $x, y \in \Re^n$
l	:	index of the entering variable
k	:	index of the leaving variable
$A_{.l}$	:	the lth column of A
$B^{-1}_{r.}$	:	the rth row of basis inverse
$\overline{B}^{-1}_{r.}$	:	the matrix B^{-1} with the rth row put to zero

This updating scheme presented by [3]. The key idea is the following. The current basis inverse $\tilde{B}^{-1}$ can be computed from the previous inverse B^{-1} with a simple outer product of two vectors and one matrix addition. Namely we have

$$\tilde{B}^{-1} = \overline{B}^{-1}_{r.} + v \otimes B^{-1}_{r.} \tag{3}$$

A formal description of this method follows.

```
Compute the pivot column $h_l = B^{-1} A_{.l}$
Compute the vector
    $v = \begin{bmatrix} -\frac{h_{1l}}{h_{rl}} & \cdots & \frac{1}{h_{rl}} & \cdots & -\frac{h_{ml}}{h_{rl}} \end{bmatrix}^T$
Compute the outer product $v \otimes B^{-1}_{r.}$
Set the rth row of $B^{-1}$ equal to zero. Save the result in $\overline{B}^{-1}_{r.}$
Compute the new basis inverse using relation (3).
```

The outer product requires m^2 multiplications and the addition of two matrices requires m^2 additions. The total cost of the above method is $2m^2$ operations (multiplications and additions). Hence, the complexity is $\Theta(m^2)$.

The computational study indicates that the modification of the product form of the inverse is faster than the product form of the inverse (Badr et al . 2005) so that we use the modification of the product form of the inverse in our implementation.

3. Computational results

The algorithm described in Section 2 have been experimentally implemented. In this section we describe our numerical experiments and present computational results, which demonstrate EPSA's efficiency on random sparse linear programs. In this computational analysis we perform a comparison between PSA and EPSA.

Our numerical experiments were performed on a PC with 2.000 MHz Pentium 4 processor, RAM 512 Mb and Windows XP operating system. Our implementation was done under the C environment.

Table 1

Computational results for sparse linear programs $n \times n$ *(CPU in seconds)*

$n \times n$	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
Density 5%					
300 × 300	4390.0	3683.9	31.7053	2090.0	18.7477
350 × 350	6139.5	4908.3	58.6021	2945.5	34.4852
400 × 400	7807.2	6276.2	97.9857	2889.7	56.9017
450 × 450	10097.8	8351.5	167.0880	3574.3	91.6215
500 × 500	12195.8	10086.5	250.6490	4212.6	134.2010
600 × 600	17546.0	14713.3	540.0090	5445.5	261.4220
700 × 700	23896.4	20107.6	1038.44 0	6930.8	463.7710
800 × 800	31213.1	31310.7	2212.4300	8708.9	775.7790
Density 10%					
300 × 300	8573.6	3370.1	31.6330	2022.9	20.0335
350 × 350	11664.1	4448.5	56.6681	2336.1	34.3671
400 × 400	15216.2	5173.4	89.0898	2473.8	53.4746
450 × 450	21208.5	7478.7	163.3977	3579.7	97.8625
500 × 500	23780.1	8424.7	272.2390	3778.9	129.3933
600 × 600	34251.0	12929.5	543.6208	5405.3	274.6750
700 × 700	46633.3	17158.6	1001.8940	6752.5	484. 2333
800 × 800	60911.2	22861.1	1965.3400	8152.7	778.2910

We run total of 280 random sparse problems of density 5% and 10%. The problems we tested were small and medium in size.

Table 2

Computational results for sparse linear programs $n \times 2n$ *(CPU in seconds)*

$n \times 2n$	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
Density 5%					
300 × 600	8787.3	5980.2	98.6748	4334.5	82.5895
350 × 700	11945.1	8599.7	197.7200	5472.4	155.1030
400 × 800	15635.8	10970.4	335.3570	6571.1	256.8260
Density 10%					
300 × 600	17117.5	5249.4	89.4705	3911.4	77.5788
350 × 700	23295.2	7000.4	170.1536	4830.2	141.2967
400 × 800	30440.6	8979.3	294.852	6290.4	231.7487

Table 3

Computational results for sparse linear programs $2n \times n$ *(CPU in seconds)*

$2n \times n$	nnz	PSA		EPSA	
		niter	CPU	niter	CPU
Density 5%					
600 ×300	8781.2	4228.3	105.287	232.75	66.0007
700× 350	12286.3	6117.2	206.368	3015.2	122.935
800 × 400	15611.6	8580.4	401.718	3675.1	193.491
Density 10%					
600 ×300	17135.2	5022.5	133.142	2464.5	75.7136
700× 350	23325.8	7058.2	278.003	3031.9	132.701
800 × 400	30468.7	9302.3	443.655	3680.5	209.991

The sparse linear optimization problems that have been solved are of the form

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

where $A \in \mathbb{R}^{m\times n}, c, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ and T denotes transposition and an intial feasible solution exists. In our computational study we use two different density cases of the problems: 5% and 10%. For each density case there are three

different dimension cases : $n\times n$, $n\times 2n$ and $2n\times n$. The first case $n\times n$, include eight different classes of problems corresponding to the values n = 300, 350, 400, 450, 500, 600, 700,800; each of these classes contains ten random sparse linear programs. Those of the other two cases $n\times 2n$ and $2n\times n$, include three different classes of problems corresponding to the values n = 300, 350, 400. The ranges of values that were used for randomly generated linear programs for all densities are $c \in$[1 500] , $A \in$[-700 1800], r = 7 and center=35.

For each size and density of the problem we bring together some statistics on the LPs used in our computational study and information on the performance of the two algorithms, PSA and EPSA. Columns of Table 1-3 contain problems size, number of non-zero elements of the constraint matrix

(excluding cost and right-hand size vector) nnz, the average number of iterations, niter and the mean CPU time, CPU.

In order to show more clearly the superiority of EPSA over PSA we provide now some tables showing for each case of density relative with the above tables.

Table 4

Ratios for sparse linear programs $n\times n$

$n\times n$	niter PSA/EPSA	CPU PSA/EPSA
Density 5%		
300 × 300	1.7626	1.6912
350 × 350	1.6664	1.6993
400 × 400	2.1719	1.7220
450 × 450	2.3365	1.8237
500 × 500	2.3944	1.8677
600 × 600	2.7019	2.0657
700 × 700	2.9012	2.2391
800 × 800	3.5953	2.8519
Mean value	2.4413	1.9951

Density 10%

300 × 300	1.6660	1.5790
350 × 350	1.9042	1.6489
400 × 400	2.0913	1.6660
450 × 450	2.0892	1.6697
500 × 500	2.2294	1.7562
600 × 600	2.3920	1.9791
700 × 700	2.5411	2.0690
800 × 800	2.8041	2.5252
Mean value	**2.2147**	**1.8616**

Table 5

Ratios for sparse linear programs $2n\times n$

$n\times n$	niter PSA/EPSA	CPU PSA/EPSA
Density 5%		
300 × 600	3.7728	1.5962
350 × 700	4.0748	1.6787
400 × 800	4.2479	2.0762
Mean value	**4.0318**	**1.7837**
Density 10%		
300 × 600	2.0379	1.7585
350 × 700	2.3280	2.0950
400 × 800	2.5275	2.1194
Mean value	**2.2978**	**1.9910**

Table 6

Ratios for sparse linear programs $n\times 2n$

$n\times n$	niter PSA/EPSA	CPU PSA/EPSA
Density 5%		
300 × 600	2.0273	1.1948
350 × 700	2.1828	1.2748
400 × 800	2.3795	1.3058
Mean value	**2.1965**	**1.2585**
Density 10%		
300 × 600	1.3421	1.1533
350 × 700	1.4493	1.2042
400 × 800	1.4275	1.2729
Mean value	**1.4063**	**1.2101**

In Tables 4-6 were present the ratios (iterations of PSA)/(iterations of EPSA) and (CPU time of PSA)/(CPU time of EPSA) for the corresponding densities and dimensions. At the end of each density case we give the mean values of the above ratios for each class of test problem. We now plot the ratios taken from Table 4 (Fig.1). The plot is for the sparse $n \times n$ liner programs for all density cases.

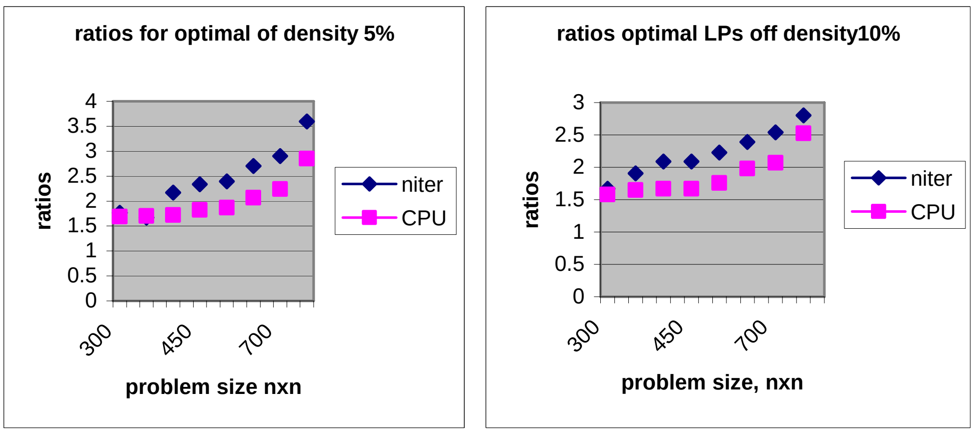

Fig.1 Ratios of PSA over EPSA for optimal sparse problems $n \times n$

From the plot above we can see that as the problem dimension increases the superiority of EPSA over PSA increases. We can see for example, in case of sparse LPs of dimension $n \times n$ that as the problem density decreases, the superiority of our algorithm over PSA increases. This conclusion is crucial because all the real life problems are extremely sparse. Particularly, in dimension 800×800 of density 5%, EPSA is 3.5953 times faster than PSA in terms of iterations and solves the problems 2.8519 times faster than PSA in terms of CPU time. Although the computational effort required in each step of EPSA requires more time compared to n iteration step of PSA, the improvement of EPSA comes from the fact that it requires adequately less iterations than PSA. Moreover, as the problem size increase and the problem density decrease, EPSA gets relatively faster.

4. Conclusion

In this paper we have resented a computational attractive approach to solving sparse linear programming problems. In the previous section we have addressed the most issue of an efficient implementation of our algorithm EPSA. The computational study of Section 3 indicates that as the problem size increases and the density of problem decreases the frequency when EPSA outperforms the simplex algorithm increases. In our opinion this is a good practical performance that gives much promise to the approach. EPSA's algorithm experimental implementation in randomly generated LPs proved that the algorithm is fast; there is still room for its further improvement. These possible improvements will be the subject of our future work.

References:

[1]Anstreich, G. ,Terlaky, T. A monotonic build-up simplex algorithm for linear programming,*Operation Research* 42 (1994) 556-561.

[2]Badr, E., Paparrizos, K., Samaras, N., Sifaleras, A. (2005) .On the basis inverse of the exterior point simplex algorithm.*17^o Εθνικό Συνέδριο Ελληνικής Εταιρίας ΕπιχειρησιακώνΕρευνών.*

[3]Benhamadou, M. (2002). On the simplex algorithm 'revised form', *Advances in Engineering Software*, 33, 769-777.

[4]Chen, H. ,Pardalos, P. Saunders, M. The simplex algorithm with a new primal and dual pivot rule, Operations Research etters 16 (1994) 121-127.

[5]Dantzig, G.B. Linear programming and extensions. *Princeton, NJ: Princeton University Press*; 1963.

[6]Dosios K., Paparrizos K., Resolution of problem of degeneracy in a primal and dul simplex algorithm, *Operation Research Letters* 20 (1997) 45-50.

[7] Karmarkar, N.K. (1984).A new polynomial-time algorithm for linear programming. Combinatorica, 4(4), 373–395.

[8] Lustig, L.J., Marsten R.E. and Shanno, D.F. (1994). Interior point methods for linear programming: computational state of the art. ORSA Journal on Computing, 6(1), 1–14.

[9]Paparrizos, K. An infeasible exterior point simplex algorithm for assignment problems,*Mathematical Programming* 51 (1991) 45-54.

[10]Paparrizos, K. A generalization of an exterior point simplex algorithm for linear programming problems, *Technical Report,University of Macedonia*, 1990.

[11]Paparrizos, K. An exterior point simplex algorithm for general linear problems, nnals of Operation Research 47 (1993) 497-508.

[12]Paparrizos, K., Samaras, N., Tsiplidis, K. Pivoting algorithm for (LP) generating two paths,in: M. P. Pardalos, A. C. Fluodas (Eds.), *Encyyclopedia of Optimization, vol 4, Kluwer Academic Publishers*, 2001, pp. 302-306.

[13]Paparrizos, K., Samaras, Stephanides G., An efficient simplex type algorithm for sparse and dense linear programs*, European Journal of Operational Research* 148 (2003) 323-334.

[14] Roos, C., Terlaky, T. and J.-Ph. Vial (1997).Theory and Algorithms for Linear Optimization. John Wiley & Sons, New York, USA.

[15]Strang, G. Introduction to applied mathematics. *Wellesley, MA: Wellesley-Cambridge Press*; 1990.

[16] Terlaky, T.and Zhang, S. (1993). Pivot rules for linear programming: a survey on recent theoretical developments. Annals of Operations Research, 46, 203–233.

[17] Wright,S.J.(1997).Primal-Dual Interior-Point Methods.SIAM, Philadelphia.