NUMERICAL MATHEMATICS
A Journal of Chinese Universities (English Series)

Vol. **15** No. 1, pp. 23-30
February 2006

Partial Pricing Rule Simplex Method with Deficient Basis†

Pingqi Pan[1,*], Wei Li[2] and Jun Cao[1]

[1] *Department of Mathematics, Southeast University, Nanjing 210096, China.*
[2] *School of Science, Hangzhou Dianzi University, Hangzhou 310018, China.*

Received June 1, 2003; Accepted (in revised version) April 12, 2004

Abstract. A new partial pricing column rule is proposed to the basis-deficiency-allowing simplex method developed by Pan. Computational results obtained with a set of small problems and a set of standard NETLIB problems show its promise of success.

Key words: Linear programming; simplex method; deficient basis; partial pricing.

AMS subject classifications: 90C05

1 Introduction

We are concerned with the linear programming problem in the standard form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0. \end{aligned} \tag{1}$$

where $A \in R^{m\times n}$ with $m < n$, $b \in R^m, c \in R^n, 1 \leq \text{rank}(A) \leq m$. It is assumed that the cost vector c, the right-hand b, and $A's$ columns and rows are nonzero, and that the equation $Ax = b$ is consistent.

It is widely accepted that pivot rules used in the simplex method affect the number of iterations required for solving linear programming problems. Much effort has therefore been made in the past on finding good pivot rules to improve the efficiency of the underlying method [1-10]. It is notable that although it usually involves more iterations, a partial pricing rule for column selection, like that used in the MINOS, involves less computational work per iteration, and consequently leads to less overall running time than the conventional full pricing.

The basis-deficiency-allowing simplex method for linear programming was developed by Pan [1] in 1997. A key feature of this method is that it uses a generalized basis allowing deficiency. Computational results obtained with this method, in which the full pricing rule is used, are

*Correspondence to: Pingqi Pan, Department of Mathematics, Southeast University, Nanjing 210096, China. Email: panpq@seu.edu.cn

†This work is supported by the NSF of China, No. 10371017 and NSF Grant of Hangzhou Dianzi University KYS091504025.

indeed very encouraging. Therefore, the following questions naturally arise: Can we introduce some partial pricing to improve the new algorithm? If so, how to do it and what is its efficiency? In this paper, we propose a new partial pricing rule for column selection for the basis-deficiency-allowing simplex method to improve its efficiency further. We report computational results obtained with a set of small problems and a set of standard NETLIB problems, and show the rule's promise of success.

2 Preliminaries

For this presentation being self-contained, we first present the basis-deficiency-allowing simplex method briefly [1]. A basis is a submatrix consisting of any $m1$ linearly independent set of A's columns, whose range space includes b. If $m1 = m$, it is a normal basis; otherwise, it is a deficient basis.

Let B be a basis with $m1$ columns and let N be nonbasis, consisting of the remaining $n - m1$ columns. Define the ordered basic and nonbasic index sets respectively by

$$J_B = \{j_1, \ldots, j_{m1}\} \text{ and } J_N = \{k_1, \ldots, k_{n-m1}\}$$

Thus we have

$$\begin{aligned} A &= [B, N] = [a_{j_1}, \ldots, a_{j_{m1}}; a_{k_1} \ldots, a_{k_{n-m1}}] \\ c^T &= [c_B^T, c_N^T] = [c_{j_1}, \ldots, c_{j_{m1}}; c_{k_1} \ldots, c_{k_{n-m1}}] \\ x^T &= [x_B^T, x_N^T] = [x_{j_1}, \ldots, x_{j_{m1}}; x_{k_1} \ldots, x_{k_{n-m1}}] \end{aligned}$$

Then program (1) can be written as

$$\begin{aligned} \min \quad & c_B^T x_B + c_N^T x_N \\ \text{s.t.} \quad & B x_B + N x_N = b \\ & x_B \geq 0,\ x_N \geq 0. \end{aligned} \tag{2}$$

Given the QR decomposition $Q^T B$=R, where $Q \in R^{m \times m}$ is orthogonal and $R \in R^{m \times m1}$ is upper triangular. Let Q and R be partitioned as $Q = [Q_1, Q_2]$ and $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, where $Q_1 \in R^{m \times m_1}, Q_2 \in R^{m \times (m-m1)}, R_1 \in R^{m_1 \times m_1}$ is upper triangular, and $0 \in R^{(m-m1) \times m_1}$ is the zero matrix. The canonical matrix may be partitioned as

$$[Q^T B \quad Q^T N \quad Q^T b] = \begin{bmatrix} R_1 & Q_1^T N & Q_1^T b \\ 0 & Q_2^T N & 0 \end{bmatrix}.$$

The associated basic solution is then

$$\bar{x}_N = 0, \quad \bar{x}_B = R_1^{-1} Q_1^T b, \tag{3}$$

the corresponding objective value is $f = c_B^T R_1^{-1} Q_1^T b$, and the corresponding reduced cost is

$$\bar{z}_N = c_N - N^T Q_1 R_1^{-T} c_B. \tag{4}$$

Let us take a single iteration. Assume that the current basic solution, say (3), is feasible, i.e., $\bar{x}_B \geq 0$. If in addition, it holds that $\bar{z}_N \geq 0$, then we are done. Suppose this is not the case. Conventionally, a nonbasic column is selected to enter the basis by Dantzig's original rule, i.e.,

$$q = \text{Argmin}\{\overline{z}_{k_j} \mid j = 1, \ldots, n - m1\} \tag{5}$$

Therefore $\overline{z}_{k_q} < 0$, and the nonbasic column a_{k_q} will enter the basis. There will be one of the following cases arising

Case 1 $m1 = m$, or otherwise $m1 < m$ but $Q_2^T a_{k_q} = 0$. In this case, a basic column can be determined to leave the basis. Introducing the $m1$-vector $\nu = R_1^{-1} Q_1^T a_{k_q}$, which may be obtained by solving the upper triangular system $R_1 v = Q_1^T a_{k_q}$, we determine a subscript p such that

$$\alpha = \frac{\bar{x}_{j_p}}{v_p} = \min\left\{\frac{\bar{x}_{j_i}}{v_i} | i \in I\right\} > 0 \tag{6}$$

where

$$I = \{i | v_i > 0, i = 1, \ldots, m1\}. \tag{7}$$

If the preceding I is empty, then the program is unbounded below. Otherwise, (6) is well defined, and the following formula can be used for updating the basic feasible solution

$$\bar{x}_B := \bar{x}_B - \alpha v \qquad \text{and} \qquad \bar{x}_{k_q} := \alpha \tag{8}$$

Case 2 $m1 < m$ and some of the $(m1+1)$ through m^{th} components of $Q^T a_{k_q}$ are nonzero. In this case, we have to append $Q^T a_{k_q}$ to the end of the basic columns, and annihilate its $(m1+2)$ through m^{th} entries by premultiplying $[QB, QN, Qb]$ by an appropriate Householder reflection.

As the number of basis' columns remains unchanged in case 1 and grows by one in case 2, the associated iterations will be referred to as rank-maintaining and rank-increasing, respectively.

Now we show how to update QR factors when column a_{k_q} is selected, and added to the end of basis B. Assume $m1 + 1 < m$, and some of the $m1 + 2$ through m^{th} components of $Q^T a_{k_q}$ is nonzero, in case 2. Determining Householder reflection $H \in R^{m \times m}$ so that the $m1 + 2$ through m^{th} components of $HQ^T a_{k_q}$ are zero, we update the QR factors as follows

$$Q^T := HQ^T \qquad \text{and} \qquad R := [R, HQ^T a_{k_q}]. \tag{9}$$

It is simpler to update QR factors after adding column a_{k_q} to the end of the basis B, in case 1. In fact, what needs to do is only adding $Q^T a_{k_q}$ to the end of R, correspondingly. Therefore, formula (9) may also be useful if matrix H there is taken to be the identity matrix for this case.

Let us show how to update QR factors after deleting the p^{th} basic column, in case 1. Denote by $\bar{R} \in R^{m \times (m1-1)}$ the matrix, resulting from dropping the p^{th} column of R. If $p = m1$, then $\bar{R}$ is itself triangular, and there is no need for any action taken. If $p < m1$, otherwise, we determine the rotations $G_j \in R^{m \times m}, j = p, \ldots, m1 - 1$, to annihilate subdiagonal entries in the p through $(m1 - 1)^{th}$ columns of the Hessenberg matrix $\bar{R}$. This leads to the updating formula below

$$Q^T := G_{m1-1} \ldots G_p Q^T \quad \text{and} \quad R := G_{m1-1} \ldots G_p \bar{R}^T. \tag{10}$$

3 The partial pricing column rule

The package MINOS, developed by Michael A. Saunders et. al in Stanford University, is accepted as one of the best implementations of the simplex algorithm for solving linear programming problems [2]. Dantzig's full pricing rule is used in MINOS for column selection for solving such problems. In addition, a 'partial pricing' tactic is also incorporated in it as an option to reduce computational work involved per iteration. This tactic first partitions the non-basic index set into a predetermined number of subsets, each of which has about an equal number of indices; then, it computes reduced costs that correspond to non-basic indices in the first subset, and

finds the smallest one among them. If the one found is not significantly small, it will do the same thing with the next subset, and so on, until a reduced cost is found small enough, and the corresponding nonbasic column is hence determined to enter the basis; or, otherwise, optimality is declared. Computational experiments show that this partial pricing rule is superior to the conventional full pricing rule practically, since it usually computes and examines a small part of reduced costs, and consequently consumes less running time overall, even though it might require more iterations. However, this tactic seems to be blind in the sense that, to a great extent, the determination of the part of reduced costs to be checked does not depend on information of reduced costs themselves. In order to overcome this disadvantage, we describe a new partial pricing rule, and incorporate it in the basic-deficiency-allowing simplex algorithm. Again define the ordered basic and non-basic index sets below

$$J_B = \{j_1, \ldots, j_m\}, \qquad J_N = \{k_1, \ldots, k_{n-m1}\}$$

For the first iteration, the determination of an entering column index is essentially the same as that under the full pricing rule. But, in order to proceed with subsequent iterations, we introduce a new index set below

$$K = \{k_j | \bar{z}_{k_j} < 0, j = 1, \ldots, n - m1\}. \tag{11}$$

If the preceding set is empty, optimality is already achieved. In the other case, we choose an entering column via

$$q = \text{Argmin}\{\bar{z}_{k_j} | k_j \in K\}. \tag{12}$$

In the next iteration, we do not have to compute all of the reduced costs by (4); instead, we only compute a part of them, i.e., those associated with the index set K

$$\bar{z}_{k_j} = c_{k_j} - a_{k_j}^T u, \qquad k_j \in K \tag{13}$$

where $u = Q_1 R_1^{-T} c_B$. Then we update K as follows

$$K := \{k_j \in K | \bar{z}_{k_j} < 0\}. \tag{14}$$

If K is nonempty, we determine q via (12). Otherwise, we have to proceed as in the first iteration by computing all of the reduced costs by (4) and redefining index set K by (11). The preceding partial pricing rule can be incorporated in the basic-deficiency-allowing simplex algorithm as follows.

Algorithm 1: Given the QR decomposition $Q^T B = R$ of an initial basis B, and the associated sets J_B, J_N . Assume that the corresponding primal solution is feasible, i.e., $\bar{x}_B$ is nonnegative.

1. Compute $\bar{z}_N$ by (4).
2. Form index set K by (11).
3. Stop if K is empty.
4. Determine column index q by rule (12).
5. Go to Step 13, if $m1 < m$ and some of the $(m1 + 1)$ through m^{th} components of $Q^T a_{k_q}$ are nonzero.
6. Compute vector $v = R_1^{-1} Q_1^T a_{k_q}$.
7. Stop if set I define by (7) is empty.
8. Determine step-length α and row index p by rule (6).
9. Update $\bar{x}$ by (8).

10. Update R and Q^T by (10).
11. Move the p^{th} index of J_B to the end of J_N .
12. Set $m1 := m1 - 1$.
13. Update R and Q^T by (9).
14. Move the q^{th} index of J_N to the end of J_B .
15. Set $m1 := m1 + 1$.
16. Compute $\bar{z}_{k_j}$ by (13).
17. Update K by (14).
18. If K is nonempty, go to Step 4; else go to Step 1.

Since there are only finitely many bases, the preceding algorithm does not terminate if and only if cycling occurs. Further, since the number of basic variables does not decrease in the process, a cycle never involves any rank-increasing iteration; or in other words, cycling can only occur with rank-maintaining iterations. If nondegeneracy is guaranteed for such iterations, therefore, there will be no chance of cycling at all, as the objective value decreases strictly. Thus, based on the well-known results in linear programming we can state the following.

Theorem 3.1. *Under the primal non-degeneracy assumption on full iterations, Algorithm* 1 *terminates at either (a): Step* 3*, with primal and dual optimal solutions reached; or (b): Step* 7*, detecting lower unboundedness of program* (1)*.*

4 Achieving primal feasibility

In order to get itself started, Algorithm 1 requires primal feasibility. In this section, we demonstrate that it is possible to achieve this by solving an auxiliary problem with piecewise-linear sums of primal infeasibilities as its objective, as have been done with the simplex method. Let us consider such a procedure to match Algorithm 1.

Assume that at the current iteration the primal deficient solution $\bar{x}$ is infeasible. Continue using the same notation as that in Section 3, we obtain

$$\bar{x}_B = R_1^{-1} Q_1^T b, \qquad \bar{x}_N = 0 \tag{15}$$

where $\bar{x}_B \ngeq 0$. If it assumed that the first l ($l \leq m1$) variables are negative, and the rest are nonnegative, then, instead of the problem (1), now the auxiliary system below should be handled:

$$\begin{aligned} \min \quad & z = \bar{c}^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{16}$$

where $-\bar{c}^T = (\overbrace{1, \ldots, 1}^{}, \overbrace{0, \ldots, 0}^{}) \in R^n$, the components of $-\bar{c}^T$ consist of l 1's and $n - l$ 0's. The following steps are then similar to those in Algorithm 1, only with row pivot rule modified.

5 Computational results

To gain some idea of the practical behavior of the proposed rule, we have performed some computational experiments. We report our computational results obtained in this section. The following two C++ codes were tested, and compared against one another:

Table 1: Comparison of two codes (Group A)

	Iterations				Total Time
	Crash	Phase 1	Phase 2	Total	
FULL	229	2	97	328	565.560
PARTIAL	229	2	106	337	543.640
FULL/PARTIAL	1.000	1.000	0.915	0.973	1.040

Table 2: Comparison of two codes (Group B)

	Iterations				Total Time
	Crash	Phase 1	Phase 2	Total	
FULL	239	5	96	340	563.360
PARTIAL	239	5	103	347	540.350
FULL/PARTIAL	1.000	1.000	0.932	0.980	1.043

1. Code PARTIAL: Algorithm 1 is used as Phase 2. A modification of it is used as Phase 1 (see Section 4).

2. Code FULL: Original basis-deficiency-allowing algorithms are implemented. Algorithm 3.3 in [1] is used as Phase-1, and Algorithm 3.1 in [1] as Phase -2.

In both codes, rows and columns of the constraint matrix are scaled first; an initial basis is then provided by Pan's dual crash procedure [1]. For the sake of numerical stability, in addition, Harris' practical row selection rule [1] is used, instead of the row selection rule (6). The difference between the two codes is that Dantzig's conventional column rule (5) is utilized in both Phase-1 and Phase-2 of FULL, whereas the partial column pivot rule, proposed in this paper, is utilized in both Phases of PARTIAL.

The first test set of problems includes two groups A and B of small problems; each group consists of 52 small problems, with sizes of from 2×7 to 15×37. The second test set includes 16 standard LP problems from NETLIB that do not have BOUNDS and RANGES sections in their MPS files.

Compiled using the VISUAL C++ 6.0; all runs were carried out under WINSOWS 98 system on a PIII 600 microcomputer, with memory 128 M bytes. The machine precision used was about 16 decimal places. The reported CPU times were measured in seconds. Considering that running time required for solving small problem was too short to measure, we arranged our computational tests so that problems in groups A and B were solved as many as 100000 times.

In Tables 1 and 2, listed are total iteration and time (by running 100000 times) for each of 52 problems with two codes, respectively. Listed are also ratios of FULL iterations to PARTIAL iterations and FULL total time to PARTIAL total time. It is seen from the tables that overall FULL required slightly fewer iterations but consumed more time than PARTIAL.

Numerical results obtained with FULL and PARTIAL for the 16 problems from NETLIB are shown in Tables 3 and 4, respectively. Table 5 compares performance of the two codes by giving iteration and total running time ratios.

It is seen from the last rows of Tables 3, 4 and 5 that total iterations by FULL and PARTIAL are 2936 and 3048, respectively. The ratio of them is 0.9630. It is also seen that total running times by two codes are 110.119 seconds and 70.235 seconds, the ratio is 1.568. So, PARTIAL requires much less running time than that of FULL overall.

Moreover, from the columns under Cols /Rows and Total Time, it is noted that PARTIAL's superiority over FULL increases with the ratio of the number of columns to that of rows for these

Table 3: Results Obtained with FULL

Problem	Rows +Cols	Iterations				Total Time	Optimal Value
		Crash	P.1	P.2	Total		
AFIRO	60	7	0	17	24	0.009	$-4.6475314286\mathrm{E}+02$
SC50B	99	5	0	57	62	0.089	$-7.0000000000\mathrm{E}+01$
SC50A	99	10	0	50	60	0.091	$-6.4575077059\mathrm{E}+01$
ADLITTLE	154	51	1	103	155	0.192	$2.2549496316\mathrm{E}+05$
BLEND	158	8	0	117	125	0.450	$-3.0812149846\mathrm{E}+01$
SHARE2B	176	76	64	54	194	1.426	$-4.1573224074\mathrm{E}+02$
SC105	209	20	0	114	134	1.764	$-5.2202061212\mathrm{E}+01$
STOCFOR1	229	77	4	64	145	2.591	$-4.1131976219\mathrm{E}+04$
SCAGR7	270	114	54	31	199	4.101	$-2.3313898243\mathrm{E}+06$
ISRAEL	317	172	0	232	404	15.308	$-8.9664482186\mathrm{E}+05$
SHARE1B	343	117	36	132	285	3.270	$-7.6589318579\mathrm{E}+04$
SC205	409	38	0	237	275	31.856	$-5.2202061212\mathrm{E}+01$
BEACONFD	436	122	0	17	139	12.770	$3.3592485807\mathrm{E}+04$
LOTFI	462	85	0	188	272	8.961	$-2.5264706062\mathrm{E}+01$
BRANDY	470	150	62	162	374	26.408	$1.5185098965\mathrm{E}+03$
SCSD1	838	5	0	83	88	0.833	$8.6666666743\mathrm{E}+00$
Total		1057	221	1658	2936	110.119	

Table 4: Results Obtained with PARTIAL

Problem	Iterations				Total Time	Optimal Value
	Crash	P.1	P.2	Total		
AFIRO	7	0	25	32	0.009	$-4.6475314283\mathrm{E}+02$
SC50B	5	0	55	60	0.065	$-7.0000000000\mathrm{E}+01$
SC50A	10	0	47	57	0.065	$-6.4575077059\mathrm{E}+01$
ADLITTLE	51	1	97	149	0.176	$2.2549496356\mathrm{E}+05$
BLEND	8	0	159	167	0.446	$-3.0812149846\mathrm{E}+01$
SHARE2B	76	68	45	189	1.004	$-4.1573224074\mathrm{E}+02$
SC105	20	0	109	129	1.673	$-5.2202061212\mathrm{E}+01$
STOCFOR1	77	4	70	151	2.526	$-4.1131976229\mathrm{E}+04$
SCAGR7	114	58	48	220	2.087	$-2.3313898243\mathrm{E}+06$
ISRAEL	172	0	212	384	8.124	$-8.9664482186\mathrm{E}+05$
SHARE1B	117	40	119	276	3.020	$-7.6589318579\mathrm{E}+04$
SC205	38	0	248	286	16.323	$-5.2202061212\mathrm{E}+01$
BEACONFD	122	0	17	139	8.281	$3.3592485807\mathrm{E}+04$
LOTFI	85	0	232	317	8.850	$-2.5264703262\mathrm{E}+01$
BRANDY	150	65	162	377	17.232	$1.5185098965\mathrm{E}+03$
SCSD1	5	0	110	115	0.464	$8.6666666758\mathrm{E}+00$
Total	1057	236	1754	3048	70.235	

NETLIB problems. This reveals that the partial pricing rule is amenable to linear programming problems with the number of columns large, relative to the number of rows. This is what we desire for the rule.

Finally, we conclude that PARTIAL outperformed FULL overall. Interesting numerical behavior of the proposed rule does show its promise of success. Further research needs to be done.

Table 5: Comparison of FULL and PARTIAL

Problem	Rows/Cols	FULL/PARTIAL			
		Iterations			
		P.1	P.2	Total	Total Time
AFIRO	1.889	-	0.680	0.750	1.000
SC50B	1.560	-	1.036	1.033	1.370
SC50A	1.560	-	1.064	1.053	1.400
ADLITTLE	2.464	1.000	1.062	1.040	1.091
BLEND	1.541	-	0.736	0.749	1.009
SHARE2B	1.688	0.941	1.200	1.026	1.420
SC105	1.552	-	1.046	1.039	1.054
STOCFOR1	1.410	1.000	0.914	.0960	1.026
SCAGR7	1.434	0.931	0.646	0.905	1.965
ISRAEL	1.816	-	1.094	1.052	1.884
SHARE1B	2.162	0.900	1.109	1.033	1.083
SC205	1.546	-	0.956	0.962	1.952
BEACONFD	1.705	-	1.000	1.000	1.542
LOTFI	2.392	-	0.810	0.861	1.013
BRANDY	1.570	0.954	1.000	0.992	1.532
SCSD1	9.870	-	0.755	0.765	1.795
Total	-	0.936	0.945	0.963	1.568

We expect that the proposed partial pricing column rule's superiority over the conventional rule increases with problem sizes.

References

[1] Harris, P M J. Pivot selection methods of the Devex LP code. Math. Prog., 1973, 5: 1-28.

[2] Bruce A. Murtagh and Michael A. Saunders MINOS 5.3 Users' Guide , Technical Report Sol 1985, Stanford University.

[3] Forrest J J H, Goddard D. Steepest - edge simplex algorithms for linear programming. Math. Prog., 1992, 57: 341-374.

[4] Harris P M J. Pivot selection methods of the Devex LP code. Math. Prog., 1973, 5: 1-28.

[5] Pan P Q. Practical finite pivoting rules for the simplex method. OR Spektrum, 1990, 12: 219-225.

[6] Pan P Q. A dual projective simplex method for linear programming. Comput. Math. Appl., 1998, 35(6): 119 -135.

[7] Pan P Q. A Basis-deficiency-allowing variation of the simplex method. Comput. Math. Appl., 1998, 36(3): 33-53.

[8] Pan P Q, Pan Y P. A phase-1 approach for the generalized simplex algorithm. Comput. Math. Appl., 2001, 42: 1455-1464.

[9] Pan P Q, Li W, Wang Y. A phase-I algorithm using the most-obtuse-angle rule for the basis-deficiency-allowing dual simplex method. OR Transaction, 2004, 8(3): 88-96.

[10] Li W. A new simplex-like algorithm for linear programming. Math. Theory Appl., 2003, 23(3): 118-122.