An International Journal
computers & mathematics with applications

PERGAMON Computers and Mathematics with Applications 42 (2001) 1455–1464
www.elsevier.nl/locate/camwa

A Phase-1 Approach for the Generalized Simplex Algorithm

PING-QI PAN
Department of Applied Mathematics
Southeast University
Nanjing, P.R. China
panpq@seu.edu.cn

YUNPENG PAN
Department of Industrial Engineering
University of Wisconsin-Madison
Madison, WI 53706, U.S.A.
pany@cae.wisc.edu

(Received and accepted July 2000)

Abstract—A new simplex variant allowing basis deficiency has recently been proposed to attack the degeneracy [1]. As a generalization of the simplex algorithm, it uses a Phase-1 procedure, solving an auxiliary problem with piecewise-linear sums of infeasibilities as its objective. In this paper, we develop another Phase-1 approach that only introduces a single artificial variable. Unlike the former, which needs a crash procedure to supply an initial basis, the proposed Phase-1 is able to get itself started from scratch, with an artificial basis having a single column. Computational results with a set of standard test problems from NETLIB are also reported.

Keywords—Degeneracy, Deficient basis, LU-decomposition, Phase-1, Single artificial variable.

1. INTRODUCTION

The simplex algorithm [2,3] has been very successful in solving linear programming (LP) problems. However, it has experienced a difficulty theoretically and practically. An iteration step could fail to take a nonzero-length step forward from a *degenerate* vertex, at which more than n hyper-planes meet in the n-space. Theoretically, this undermines the finiteness of the simplex algorithm; practically, this could lead to its stalling for too long a time before exiting a vertex, and consequently degrading efficiency, or even failing to solve a large problem completely.

In our opinion, the simplex algorithm should be better to be viewed as a basis-searching rather than vertex-searching approach. In fact, it proceeds to a new basis in each iteration step, different

The first author of this work was partially supported by the National Natural Science Foundation of China, No. 19971014.
The first author would like to thank Professor George B. Dantzig for his enthusiastic encouragement and advice, given during and after the 16th International Symposium on Mathematical Programming held in Lausanne, Switzerland, August 24–29, 1997.

0898-1221/01/$ - see front matter Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
PII: S0898-1221(01)00254-1

from the old basis by a single column. Unfortunately, it might not move to a new vertex since in the simplex methodology, more than one basis can correspond to a single vertex if it is degenerate, and no pivot rule has guaranteed to be able to specify the next basis that corresponds to a really different vertex in this case.

Therefore, it seems to be attractive to establish a one-to-one correspondence between vertices and bases, or at lease, to make as few bases as possible correspond to a single vertex. To this end, it is logical to modify the square basis concept by allowing its deficiency, characterized as one having fewer columns than rows.

Along this line, a generalization of the simplex algorithm using the QR-decomposition was proposed [4]. Recently, an LU-decomposition-based variant was made [1]. The Phase-1 of the latter starts with an initial generalized basis, provided by a crash heuristic, and solves an auxiliary program with piecewise-linear sums of infeasibilities as its objective to achieve feasibility. The purpose of writing this paper is to present a new Phase-1 approach that can get itself started from scratch with an artificial basis having only a single column.

We are concerned with the LP problem in the following standard form:

$$\min \quad c^\top x, \tag{1.1a}$$

$$\text{s.t.} \quad Ax = b, \qquad x \geq 0, \tag{1.1b}$$

where $A \in \mathcal{R}^{m \times n}$ with $m < n$, and $b \in \mathcal{R}^m$, $c \in \mathcal{R}^n$. It is assumed that the cost vector c, the right-hand side b, and A's columns and rows are nonzero, and that $Ax = b$ is consistent. No assumption is made on the rank of A, except $1 \leq \text{rank}(A) \leq m$.

The following notation will be utilized throughout this paper:

a_j	the j^{th} column of A,
$a_{i,j}$	the i^{th} row and j^{th} column entry of A,
e_i	the unit vector with the i^{th} component 1,
$\bullet_j$	the j^{th} component of a vector $\bullet$,
$\| \bullet \|$	the 2-norm of a vector $\bullet$,
$R(\bullet)$	the range space of a matrix $\bullet$.

In the next section, for self-contained the generalization of the simplex algorithm is first presented. Then in Section 3, the Phase-1 approach is established. Finally, computational results, obtained with a set of standard test problems from NETLIB, are reported in Section 4.

2. THE GENERALIZATION OF THE SIMPLEX ALGORITHM

In this section, we first generalize the basis concept and then describe a variant of the simplex algorithm using such a basis.

Basis is defined as a *square* nonsingular submatrix from the coefficient matrix, of order exactly equal to the number of rows of the coefficient matrix. Since the number of basis' columns is fixed to m, some basic variables must have value zero, whenever the right-hand side belongs to a *proper* subspace of the range space of the basis. Therefore, a direct way to ease degeneracy is to remove those basic columns that do not belong to the proper subspace.

The preceding idea leads to the following redefinition of the basis for $Ax = b$, or any system equivalent to it.

DEFINITION 2.1. *A basis is a submatrix consisting of any linearly independent set of columns of the coefficient matrix, whose range space includes the right-hand side.*

All bases now fall into the following two categories.

DEFINITION 2.2. *If the number of columns of a basis equals the number of rows of the coefficient matrix, it is a full basis; else, it is a deficient one.*

Clearly, the simplex algorithm merely uses the full basis. We shall demonstrate how to modify it using the generalized basis, just introduced above.

Let B be a basis with s columns and let N be *nonbasis*, consisting of the remaining columns of A. Define the ordered *basic* and *nonbasic* index sets, respectively, by

$$J_B = \{j_1, \ldots, j_s\} \qquad \text{and} \qquad J_N = \{k_1, \ldots, k_{n-s}\}, \tag{2.1}$$

where j_i, $i = 1, \ldots, s$, is the index of the i^{th} column of B, and k_j, $j = 1, \ldots, n-s$, the index of the j^{th} column of N. The subscript i of a basic index j_i is called *row index*, and the subscript j of a nonbasic index k_j *column index*. Components and columns of associated vectors and matrices, corresponding to basic and nonbasic indices, are called *basic* and *nonbasic*, respectively. For simplicity of exposition, hereafter, components of vectors and columns of matrices will always be arranged, and partitioned conformably, as the ordered set $\{J_B, J_N\}$ changes. For instance, we have

$$\begin{aligned} A &= [B, N] = \left[a_{j_1}, \ldots, a_{j_s}; a_{k_1}, \ldots, a_{k_{n-s}}\right], \\ c^\top &= \left[c_B^\top, c_N^\top\right] = \left[c_{j_1}, \ldots, c_{j_s}; c_{k_1}, \ldots, c_{k_{n-s}}\right], \\ x^\top &= \left[x_B^\top; x_N^\top\right] = \left[x_{j_1}, \ldots, x_{j_s}; x_{k_1}, \ldots, x_{k_{n-s}}\right]. \end{aligned}$$

Using the preceding notation, program (1.1) can be written

$$\begin{aligned} \min \quad & f = c_B^\top x_B + c_N^\top x_N, && \text{(2.2a)} \\ \text{s.t.} \quad & Bx_B + Nx_N = b, && \text{(2.2b)} \\ & x_B \geq 0, \qquad x_N \geq 0. && \text{(2.2c)} \end{aligned}$$

Now form the initial tableau below:

$$\begin{bmatrix} B & N & b \\ c_B^\top & c_N^\top & 0 \end{bmatrix}. \tag{2.3}$$

The preceding will be continuously modified by premultiplying it by Gauss transformations with row exchanges until an optimal one is reached (see below). Clearly, all such tableaus are equivalent to one another, in the sense of their representing the program (2.2), or (1.1).

Without loss of generality, hereafter we shall assume that the number of basic columns is less than the number of rows, i.e., $s < m$. Suppose that we are currently faced with the following tableau, with the associated sets J_B and J_N given, as partitioned:

$$\begin{bmatrix} U & \bar{N} & \bar{b} \\ 0 & \bar{z}_N^\top & -f \end{bmatrix} \equiv \begin{bmatrix} U_1 & \bar{N}_1 & \bar{b}_1 \\ 0 & \bar{N}_2 & 0 \\ 0 & \bar{z}_N^\top & -f \end{bmatrix}, \tag{2.4}$$

where $U_1 \in \mathcal{R}^{s \times s}$ is upper triangular. A tableau with upper triangular basic columns, like (2.4), is termed a *canonical* tableau. Since the $(s+1)$ through m^{th} components of $\bar{b}$ are zero, the upper triangular matrix $U \in \mathcal{R}^{m \times s}$ is a basis. The related basic solution is then

$$\begin{aligned} \bar{x}_N &= 0, && \text{(2.5a)} \\ \bar{x}_B &= U_1^{-1} \bar{b}_1, && \text{(2.5b)} \end{aligned}$$

corresponding to the objective value f.

It is easy to show that if $\bar{x}_B \geq 0$ and $\bar{z}_N \geq 0$, then an optimal basic solution is already obtained, and hence, we are done. Such a tableau is termed *optimal*. Now assume that this is not the case but tableau (2.4) is feasible, i.e., $\bar{x}_B \geq 0$ and $\bar{z}_N \ngeq 0$. Take notation

$$\bar{A} = \left[U, \bar{N}\right] = \left[\bar{a}_{j_1}, \ldots, \bar{a}_{j_s}, \bar{a}_{k_1}, \ldots, \bar{a}_{k_{n-s}}\right].$$

To make a basis change, we select a nonbasic column to enter the basis under some column selection criteria, for instance, the following analogy to Dantzig's original one:

$$q = \text{Arg}\min\left\{\bar{z}_{k_j} \mid j = 1, \ldots, n-s\right\}. \tag{2.6}$$

Thus, $\bar{z}_{k_q} < 0$, and the nonbasic column $\bar{a}_{k_q}$ will enter the basis. There will be one of the following two cases arising, which need to be treated separately.

CASE 1. $s = m$, or $s < m$ but all the $(s+1)$ through m^{th} entries of $\bar{a}_{k_q}$ are zero. This case occurs if and only if $\bar{a}_{k_q} \in R(U)$.

In this case, the value of the $\bar{x}_{k_q}$ is allowed to increase from zero, with the objective value decreasing. Denote by $(\bullet)^{(s)}$ the subvector consisting of the first s components of a vector $\bullet$. To determine a blocking basic variable, compute the s-vector

$$v = U_1^{-1}(\bar{a}_{k_q})^{(s)}, \tag{2.7}$$

by solving the upper triangular system $U_1 v = (\bar{a}_{k_q})^{(s)}$. There is no blocking variable, and the program is hence, unbounded below if $v \leq 0$. In the other case, a row index p can be determined such that

$$\alpha = \bar{x}_{j_p}/v_p = \min\left\{\bar{x}_{j_i}/v_i \mid v_i > 0,\ i = 1, \ldots, s\right\} > 0, \tag{2.8}$$

where the inequality holds under nondegeneracy, i.e., $\bar{x}_B > 0$. Therefore, as the value of $\bar{x}_{k_q}$ grows from zero up to α with values of the other nonbasic components of the basic solution fixed on zero, the value of $\bar{x}_{j_p}$ decreases down to zero, subject to primal feasibility. This leads to the following formula for updating the basic feasible solution $\bar{x}$:

$$\bar{x}_{j_i} := \bar{x}_{j_i} - \alpha v_i, \qquad \forall\, i = 1, \ldots, s; \qquad \bar{x}_{k_q} := \alpha. \tag{2.9}$$

Effects of a basis change is to bring the p^{th} basic column of the canonical tableau to the end of its nonbasic columns, with J_B and J_N adjusted conformably. If $p = s$, the resulting $U \in \mathcal{R}^{m\times(s-1)}$ is already upper triangular, while when $p < s$, it is an upper Hessenberg with nonzero subdiagonal entries in its p through $(s-1)^{\text{th}}$ columns. In the latter case, the unwanted entries are zeroed via Gaussian elimination, with possible row exchanges for larger diagonal pivots. Then, the q^{th} nonbasic column of (2.4) is brought to the end of its basic columns, with J_B and J_N adjusted conformably. It is easy to show that the s^{th} entry of the newly-entering basic column is nonzero, and hence, the resulting $U \in \mathcal{R}^{m\times s}$, or $U_1 \in \mathcal{R}^{s\times s}$ is again upper triangular, with nonzero diagonal entries.

CASE 2. $s < m$ and some of the $(s+1)$ through m^{th} entries of $\bar{a}_{k_q}$ are nonzero. This case occurs if and only if $\bar{a}_{k_q} \notin R(U)$.

This time, the value of $\bar{x}_{k_q}$ is not allowed to increase from zero because this will lead to the violation of some of the last $m-s$ constraints. We are, therefore, forced to increase the number of basis' columns. Set $s := s+1$. Annihilate the $(s+1)$ through m^{th} entries, if any, of $\bar{a}_{k_q}$ via Gaussian elimination with a row exchange for a largest possible diagonal pivot, and bring the q^{th} nonbasic column of canonical tableau (2.4) to the end of its basic part. After that, we rearrange J_B and J_N conformably. Clearly, the resulting $U \in \mathcal{R}^{m\times s}$ is upper triangular, with nonzero diagonal entries.

It is observed that the Gaussian elimination carried out in either case do not disturb $(s+1)$ through m^{th} zero components of the right-hand side at all. Thus, a new canonical tableau presents, once the newly-entering basic entry of the cost row is zeroed via Gaussian elimination. The description of a single iteration is then completed. Such steps are repeated until optimality is accomplished, or otherwise lower unboundedness is detected. It is noted that the number of basis' columns varies dynamically, but never decreases, in the solution process.

Since the number of basis' columns remains unchanged in Case 1, and grows by one in Case 2, the corresponding iterations will be referred to as *full* and *rank-increasing* ones, respectively.

The preceding steps are summarized into the following model.

ALGORITHM 2.3. *Main procedure: tableau version. Let (2.4) be an initial canonical tableau, and let J_B and J_N be the associated index sets. Given the basic feasible solution $\bar{x}$, featured by (2.5).*

1. *Stop if $\bar{z}_N \geq 0$.*
2. *Determine a column index q by rule (2.6).*
3. *If $s < m$ and some of the $(s+1)$ through m^{th} entries of $\bar{a}_{k_q}$ are nonzero, do the following:*
   (i) set $s := s + 1$,
   (ii) determine a row index r such that $\|\bar{a}_{r.k_q}\| = \max\{\|\bar{a}_{i.k_q}\| \mid i = s, \ldots, m\}$,
   (iii) swap the s and r^{th} rows if $r \neq s$,
   (iv) zero the $(s+1)$ through m^{th} entries of $\bar{a}_{k_q}$ using Gaussian elimination,
   (v) go to Step 10.
4. *Compute vector v, defined by (2.7).*
5. *Stop if $v \leq 0$.*
6. *Determine a row index p and a step-length α by (2.8).*
7. *Updated $\bar{x}$ by (2.9).*
8. *Bring the p^{th} basic column of the tableau to the end of its nonbasic part, and adjust sets J_B and J_N conformably.*
9. *If $p < s$, then for $k = p, \ldots, s-1$ do the following:*
   (i) determine a row index r such that $\|\bar{a}_{r.j_k}\| = \max\{\|\bar{a}_{i.j_k}\| \mid i = k, k+1\}$;
   (ii) swap the k and r^{th} rows if $r \neq k$;
   (iii) zero the sub-diagonal entry of $\bar{a}_{j_k}$ via Gaussian elimination.
10. *Bring the q nonbasic column of the tableau to the end of its basic part, and adjust J_B and J_N conformably.*
11. *Zero $\bar{z}_{j_s}$ via Gaussian elimination.*
12. *Go to Step 1.*

Since there are only finitely many bases, the algorithm does not terminate if and only if cycling occurs. Furthermore, since the number of columns of a basis never decreases in the process, a cycle never involves any rank-increasing iteration. In other words, cycling can only occur in full iterations. If nondegeneracy is assumed for full iterations, there will be no chance of cycling at all, since the objective value decreases strictly. Thus, based on the discussions made prior to the introduction of the algorithm, we state the following.

THEOREM 2.4. *Under nondegeneracy assumption on all full iterations, Algorithm 2.3 terminates at either*

(1) *Step 1, with an optimal basic solution reached; or*
(2) *Step 5, detecting lower unboundedness of program (1.1).*

It is also possible to derive a revised version of Algorithm 2.3 [1].

3. THE PHASE-1 PROCEDURE

Algorithm 2.3 needs a feasible basis to get itself started. For this purpose, in this section we develop a Phase-1 approach which starts with an artificial basis having a single column.

Introduce an artificial variable x_{n+1}, and construct the following auxiliary program:

$$\begin{aligned} \min \quad & x_{n+1}, && \text{(3.1a)} \\ \text{s.t.} \quad & Ax + x_{n+1} b = b, && \text{(3.1b)} \\ & x \geq 0, \qquad x_{n+1} \geq 0. && \text{(3.1c)} \end{aligned}$$

We have the following results concerning the preceding program.

THEOREM 3.1. *Program (3.1) has an optimal solution with a nonnegative optimal value. Moreover, the feasible region of the original program (1.1) is nonempty if and only if the optimal value of (3.1) is equal to zero.*

PROOF. Set index sets as follows:

$$J_B = \{j_1\} = \{n+1\} \qquad \text{and} \qquad J_N = \{k_1, \ldots, k_n\} = \{1, \ldots, n\}, \tag{3.2}$$

and correspondingly set nonbasis $N = A$, and basis $B = (b)$. Then we have the associated basic solution to (3.1), i.e.,

$$\bar{x} = 0 \qquad \text{and} \qquad \bar{x}_{n+1} = 1, \tag{3.3}$$

which is clearly feasible. Seeing that $x_{n+1} \geq 0$, therefore, we can assert that program (3.1) has an optimal solution with an nonnegative optimal value. Let $\begin{pmatrix} \bar{x} \\ \bar{x}_{n+1} \end{pmatrix}$ be an optimal solution to (3.1). If $\bar{x}_{n+1} = 0$, the $\bar{x}$ is clearly a feasible solution to (1.1). Conversely, if $\bar{x}$ is a feasible solution to (1.1), then $\begin{pmatrix} \bar{x} \\ 0 \end{pmatrix}$ is an optimal solution to (3.1). That is to say, there exists a feasible solution to program (1.1) if and only if the optimal value of program (3.1) is equal to zero. ∎

Therefore, our Phase-1 is merely to carry out the procedure, described in the foregoing section, to solve (3.1), starting with initial sets J_B and J_N, defined by (3.2), and the corresponding initial basis $B = (b)$ and nonbasis $N = A$.

Such Phase-1 procedure has two attractive features. First, it is simper than the Phase-1 by solving an auxiliary program with piecewise-linear sums of infeasibilities as its objective, or other variants of it. Second, it can get itself started from scratch, and hence, there will be no need for any crash procedure to provide initial basis and nonbasis.

4. COMPUTATIONAL EXPERIMENTS

In order to gain an insight into the behavior of the proposed approach, we have performed some computational experiments (without exploiting the sparseness).

The following two FORTRAN 77 modules were tested, and compared.

Code RSA: A conventional implementation of the revised two-phase simplex algorithm, in which the inverse of the basis is updated explicitly in each iteration step.

Code PAG: Algorithm 2.3 serves as its Phase-2 procedure. It is also used in Phase-1 to solve program (3.1) to achieve feasibility.

In all runs reported below, the problems were first reduced in size by a preprocessor to remove redundant rows before executing RSA, whereas no such action was needed by PAG; the rows and columns of the constraint matrix were scaled by the preprocessor for both codes. Harris' pivot strategy [5] was incorporated into each code fittingly.

Compiled using the NDP-FORTRAN-386 VER. 2.1.0. with default options, all runs were carried out under DOS 6.2 system on an IBM 486/66 DX2 compatible microcomputer, with memory 32 Mbytes available. The machine precision used was about 16 decimal places. Pivot tolerance taken was 10^{-8}, and both the primal and dual feasibility tolerance were 10^{-6}. The reported CPU times were measured in seconds with utility routine DOSTIM. However, the time required by the preprocessor was not included.

The test set of problems includes 25 standard LP problems from NETLIB that do not have BOUNDS and RANGES sections in their MPS files [6] since the current version of our codes cannot handle them implicitly. As the largest possible subset of NETLIB problems of such kind that can be solved in our computing environment, they are the first 25 problems in the order of increasing sum of the numbers of rows and columns in the coefficient matrix, before adding slack variables.

Numerical results obtained with RSA are shown in Table 4.1, where numbers of rows and columns of each tested problem is listed in the columns labeled M and N, and their sum in the

Table 4.1. Code RSA statistics.

Problem	M	N	$M+N$	Total			Phase-1		
				Iter	Time	% Dgn	Iter	Time	% Dgn
AFIRO	27	32	59	29	0.22	68.97	28	0.22	71.43
SC50B	50	48	98	59	1.32	77.97	51	1.16	88.24
SC50A	50	48	98	57	1.27	71.93	51	1.16	78.43
ADLITTLE	56	97	153	128	4.07	13.28	69	2.31	24.64
BLEND	74	83	157	115	6.10	39.13	90	4.89	50.00
SHARE2B	96	79	175	196	16.53	57.14	149	12.85	67.79
SC105	105	103	208	123	13.51	70.73	110	12.25	77.27
STOCFOR1	117	111	228	174	23.62	70.11	150	20.65	80.67
SCAGR7	129	140	269	181	31.31	33.70	146	25.82	41.78
ISRAEL	174	142	316	513	159.73	0.78	188	64.16	1.06
SHARE1B	117	225	342	309	49.87	7.12	190	31.86	11.58
SC205	205	203	408	262	133.36	67.56	211	109.74	81.04
BEACONFD	173	262	435	213	81.45	51.17	159	62.45	57.86
LOTFI	153	308	461	348	106.23	13.51	190	60.64	22.63
BRANDY	220	249	469	356	164.34	34.27	238	113.59	50.00
E226	223	282	505	615	396.07	22.60	394	262.93	32.49
AGG	488	163	651	640	2002.53	7.03	572	1814.74	6.99
SCORPION	388	358	746	404	742.15	60.89	380	701.84	64.21
BANDM	305	472	777	674	929.56	22.85	489	688.93	30.06
SCTAP1	300	480	780	480	653.78	35.21	344	479.61	43.60
SCFXM1	330	457	787	595	962.35	35.46	429	710.35	46.39
AGG2	516	302	818	823	3018.92	4.86	629	2358.44	5.56
AGG3	516	302	818	839	3072.64	5.60	627	2352.02	6.22
SCSD1	77	760	837	195	35.15	77.44	80	15.10	97.50
SCAGR25	471	500	971	911	2935.06	30.74	616	2042.13	40.26
TOTAL	5360	6206	11566	9239	15541.14	27.21	6580	11949.84	34.83

column labeled $M+N$. Total iterations and time, required for solving each problem, are displayed in the two columns labeled *Iter* and *Time* under *Total*, and those required by Phase-1 in the two columns under *Phase-1*; the percentage of degenerate iterations are given in the columns labeled *% Dgn*. Results associated with PAG are given in Table 4.2. All runs terminated with correct optimal objective values reached, as those in NETLIB index file.

Table 4.3 offers a comparison of RSA vs. PAG. Ratios of RSA total iterations and time to PAG total iterations and time are, respectively, given in the two columns labeled *Iter* and *Time* under *Total*, while ratios associated with Phase-1 are listed under *Phase-1*. The line labeled *Total* indicates that time ratios for total and Phase-1 are as high as 8.27 and 11.17, respectively. Thus, PAG unambiguously outperforms RSA without any exception on all the test problems. We would like to point out that PAG is also slightly more efficient than GSA, which are dealt with in the earlier paper [1].

To see how the method performs with the increase of sizes of test problems, we divide the 25 problems into three groups: the first named "SMALL" includes the first eight problems, from

Table 4.2. Code PAG statistics.

Problem	Total						Phase-1		
	Iter	Time	% Dgn	% Dfc	% M1	% Nfl	Iter	Time	% Dgn
AFIRO	31	0.06	3.23	100.00	96.30	16.13	8	0.00	0.00
SC50B	61	0.17	0.00	100.00	96.00	21.31	6	0.06	0.00
SC50A	60	0.16	0.00	100.00	98.00	18.33	11	0.05	0.00
ADLITTLE	148	1.87	8.11	52.03	100.00	62.16	50	0.38	8.00
BLEND	123	1.26	14.63	62.60	100.00	39.84	9	0.05	0.00
SHARE2B	195	3.96	33.85	97.44	100.00	50.77	139	3.13	44.60
SC105	132	1.82	3.03	100.00	98.10	21.97	21	0.22	0.00
STOCFOR1	137	2.48	2.19	94.16	100.00	14.60	85	1.38	0.00
SCAGR7	198	9.77	13.13	80.30	100.00	34.85	152	5.82	17.11
ISRAEL	444	27.95	0.00	81.31	100.00	60.81	172	7.74	0.00
SHARE1B	313	19.77	14.38	42.49	100.00	62.62	163	10.71	27.61
SC205	294	20.43	9.18	100.00	98.54	31.29	39	0.99	0.00
BEACONFD	141	10.60	0.00	100.00	70.52	13.48	123	7.91	0.00
LOTFI	305	28.28	16.72	87.87	100.00	49.84	86	4.01	0.00
BRANDY	314	34.00	9.24	100.00	88.08	45.86	191	17.90	15.18
E226	669	76.35	9.42	100.00	94.17	68.61	157	16.76	9.55
AGG	547	130.40	4.57	100.00	98.36	12.25	473	110.35	3.38
SCORPION	377	64.38	1.06	100.00	95.52	9.55	290	50.92	1.38
BANDM	538	158.90	7.43	100.00	98.69	44.05	314	77.50	8.28
SCTAP1	368	81.29	12.50	100.00	89.67	26.90	249	50.80	9.64
SCFXM1	511	157.31	6.65	100.00	99.09	36.01	299	69.54	8.03
AGG2	693	294.18	4.18	96.39	100.00	25.54	516	172.08	0.00
AGG3	701	301.48	4.42	96.01	100.00	26.39	516	172.02	0.00
SCSD1	121	11.21	32.23	81.82	100.00	36.36	6	0.28	0.00
SCAGR25	693	440.06	14.14	100.00	98.30	33.19	555	288.96	17.12
TOTAL	8114	1878.14	8.52	93.30	96.87	36.70	4630	1069.56	7.99

AFIRO to STOCFOR1, the second named "MEDIUM" includes the following eight problems, from SCAGR7 to E226, and the third named "LARGE" consists of the last nine problems, from AGG to SCAGR25. Ratios for each group as a whole are given in the bottom three lines of Table 4.3. It is seen that PAG's superiority over RSA grows with the increase of problem sizes, overall.

Nevertheless, we do not want to claim too much about the superiority of the new approach over modern simplex implementations based on our computational tests done at this stage. Rather, what we want to deal with here is PAG's distinctive and favorable behavior that makes itself such an unambiguous winner compared with RSA.

The first gain is bounded up with the reduction of effects of degeneracy. In Tables 4.1 and 4.2, the columns labeled *% Dgn* reveal that overall, for RSA, 27.21% of total iterations and 34.83% of Phase-1 iterations are degenerate (with zero-length steps), while, for PAG, only 8.52% of total iterations and 7.99% of Phase-1 iterations are such ones.

Another benefit stems from the large number of deficient bases encountered in PAG's solution process. In Table 4.3, the column labeled *% Dfc* indicates that as high as 93.30% of total bases

Table 4.3. Ratio of RSA to PAG.

Problem	$M+N$	Total		Phase-1	
		Iter	Time	Iter	Phase-1
AFIRO	59	0.94	3.67	3.50	–
SC50B	98	0.97	7.76	8.50	19.33
SC50A	98	0.95	7.94	4.64	23.20
ADLITTLE	153	0.86	2.18	1.38	6.08
BLEND	157	0.93	4.84	10.00	97.80
SHARE2B	175	1.01	4.17	1.07	4.11
SC105	208	0.93	7.42	5.24	55.68
STOCFOR1	228	1.27	9.52	1.76	14.96
SCAGR7	269	0.91	3.20	0.96	4.44
ISRAEL	316	1.16	5.71	1.09	8.29
SHARE1B	342	0.99	2.52	1.17	2.97
SC205	408	0.89	6.53	5.41	110.85
BEACONFD	435	1.51	7.68	1.29	7.90
LOTFI	461	1.14	3.76	2.21	15.12
BRANDY	469	1.13	4.83	1.25	6.35
E226	505	0.92	5.19	2.51	15.69
AGG	651	1.17	15.36	1.21	16.45
SCORPION	746	1.07	11.53	1.31	13.78
BANDM	777	1.25	5.85	1.56	8.89
SCTAP1	780	1.30	8.04	1.38	9.44
SCFXM1	787	1.16	6.12	1.43	10.21
AGG2	818	1.19	10.26	1.22	13.71
AGG3	818	1.20	10.19	1.22	13.67
SCSD1	837	1.61	3.14	13.33	53.93
SCAGR25	971	1.31	6.67	1.11	7.07
TOTAL	11566	1.14	8.27	1.42	11.17
SMALL	1176	0.99	5.66	2.12	10.53
MEDIUM	3205	1.04	4.94	1.58	10.18
LARGE	7185	1.22	8.76	1.29	11.25

are deficient overall. In fact, the column labeled *% M1*, giving the percentage of the number of columns of the final basis reached to the number of rows, indicates that a *deficient* optimal basis is obtained with more than a half of the test problems (14 out of the 25).

Another profit is related to the fact that rank-increasing iterations constitute the majority. Recall that while such an iteration appends a column to the basis, a full iteration not only appends to but also drops from the basis' columns. In Table 4.2, the column labeled *% Nfl* reveals that the number of full iterations is low. Overall, only a little more than one third of total iterations are full (36.70%); that is to say, nearly two thirds of all the iterations are rank-increasing ones. This should be a good indication of the high efficiency of the new code.

REFERENCES

1. P.-Q. Pan, Generalizing the simplex algorithm to the degenerate case (to appear).
2. G.B. Dantzig, Programming in a linear structure. Comptroller, USAF, Washington, DC, (February, 1948).
3. G.B. Dantzig, Programming of interdependent activities, II, mathematical model. In *Activity Analysis of Production and Allocation*, (Edited by T.C. Koopmans), pp. 19–32, John Wiley & Sons, New York, (1951); *Ecomometrica*, **17** (3/4), 200–211, (1949).

4. P.-Q. Pan, A basis-deficiency-allowing variation of the simplex method, *Computers Math. Applic.* **36** (3), 33–53, (1998).
5. P.M.J. Harris, Pivot selection methods of the Devex LP code, *Mathematical Programming Study* **4**, 30–57, (1975).
6. D.M. Gay, Electronic mail distribution of linear programming test problems, *Mathematical Programming Society COAL Newsletter* **13**, 10–12, (1985).
7. R.H. Bartels, A stabilization of the simplex method, *Num. Math.* **16**, 414–434, (1971).
8. R.H. Bartels and G.H. Golub, The simplex method of linear programming using LU decomposition, *Communication ACM* **12**, 275–278, (1969).
9. M. Benichou, J.M. Gauthier, G. Hentges and G. Ribiere, The efficient solution of linear programming problems—Some algorithmic techniques and computational results, *Mathematical Programming* **13**, 280–322, (1977).
10. R.G. Bland, New finite pivoting rules for the simplex method, *Mathematics of Operations Research* **2**, 103–107, (1977).
11. E.M.L. Beale, Cycling in the dual simplex algorithm, *Naval Research Logistics Quarterly* **2**, 269–275, (1955).
12. A. Charnes, Optimality and degeneracy in linear programming, *Econometrica* **20**, 160–170, (1952).
13. V. Chvatal, *Linear Programming*, Freeman and Company, San Francisco, (1983).
14. G.B. Dantzig, A. Orden and P. Wolfe, The generalized simplex method for minimizing a linear form under linear inequality restraints, *Pacific Journal of Mathematics* **5**, 183–195, (1955).
15. G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, (1963).
16. J.J.H. Forrest and D. Goldfarb, Steepest-edge simplex algorithms for linear programming, *Mathematical Programming* **57**, 341–374, (1992).
17. K. Fukuda and T. Terlaky, Criss-cross methods: A fresh view on pivot algorithms, *Mathematical Programming, (Series B)* **79**, 369–396, (1997).
18. P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright, A practical anti-cycling procedure for linearly constrained optimization, *Mathematical Programming* **45**, 437–474, (1989).
19. D. Goldfarb and M.J. Todd, Linear programming, in optimization, In *Handbook in Operations Research and Management Science, Volume 1*, (Edited by G.L. Nemhauser and A.H.G. Rinnooy Kan), pp. 73–170, (1989).
20. G.H. Golub and C.F. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, MD, (1983).
21. H.-J. Greenberg, Pivot selection tactics, In *Design and Implementation of Optimization Software*, (Edited by H.J. Greenberg), pp. 109–143, Sijthoff and Noordhoff, (1978).
22. A.J. Hoffman, Cycling in the simplex algorithm, Report No. 2974, Nat. Bur. Standards, Washington, DC, (1953).
23. W. Orchard-Hayes, Background development and extensions of the revised simplex method, Rand Report RM 1433, The Rand Corporation, Santa Monica, CA, (1954).
24. P.-Q. Pan, Practical finite pivoting rules for the simplex method, *OR Spektrum* **12**, 219–225, (1990).
25. P.-Q. Pan, A simplex-like method with bisection for linear programming, *Optimization* **22** (5), 717–743, (1991).
26. P.-Q. Pan, A modified bisection simplex method for linear programming, *Journal of Computational Mathematics* **14** (3), 249–255, (1996).
27. P.-Q. Pan, The most-obtuse-angle row pivot rule for achieving dual feasibility: A computational study, *European Journal of Operations Research* **101**, 167–176, (1997).
28. P.-Q. Pan, A dual projective simplex method for linear programming, *Computers Math. Applic.* **35** (6), 119–135, (1998).
29. P.-Q. Pan, A projective simplex method for linear programming, *Linear Algebra and Its Applications* **292**, 99–125, (1999).
30. P.-Q. Pan, A projective simplex algorithm using LU decomposition, *Computers Math. Applic.* **39** (1/2), 187–208, (2000).
31. A. Saunders, The complexity of LU-factorization for large-scale linear programming, In *The Complexity of Computational Problem Solving*, (Edited by R.S. Anderssen and R.P. Brent), pp. 214–230, Queensland University Press, Queensland, (1976).
32. A. Schrijver, *The Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester, (1986).
33. Scicon, *Sciconic/VM User Guide, Version 1.40*, Scicon Limited, Milton Keynes, (1986).
34. T. Terlaky, A convergent criss-cross method, *Math. Oper. und Stat. Ser. Optimization* **16** (5), 683–690, (1985).
35. S. Zionts, The criss-cross method for solving linear programming problems, *Management Science* **15** (7), 426–445, (1969).