# A NONSTANDARD SIMPLEX ALGORITHM FOR LINEAR PROGRAMMING

PING-QI PAN*

**Abstract.** The simplex algorithm travels, on the underlying polyhedron, from vertex to vertex until reaching an optimal vertex. With the same simplex framework, the proposed algorithm generates a series of feasible points (which are not necessarily vertices). In particular, it is exactly an interior point algorithm if the initial point used is interior. Computational experiments show that the algorithm are very efficient, relative to the standard simplex algorithm. It terminates at an approximate optimal vertex, or at an optimal vertex if a simple purification is incorporated.

**Key words.** simplex algorithm, nonstandard, feasible point, column rule, purification

**AMS subject classifications.** 65K05, 90C05

**1. Introduction.** Consider the linear programming (LP) problem in the following form:

$$\begin{aligned} \min \quad & c^T x \\ \text{s. t.} \quad & Ax = b, \quad l \le x \le u, \end{aligned} \tag{1.1}$$

where $A \in R^{m\times n}(m < n)$; $\mathrm{rank}(A) = m$. It is assumed that the values of components of $l$ and $u$ are finite. For problems with infinite bounds, one can use very small negative and/or very large positive numbers instead.

The simplex algorithm for solving LP problems has been very successful in practice since 1947 when George B. Dantzig founded it [10]. Recognized as one of The Ten Algorithms in the Twenty Century (IEEE2002), it is one of the most famous and widely used mathematical tools in the world.

William Orchard-Hays's contribution to this field have played an important role in the simplex algorithm's success. In 1954, he first implemented the revised simplex algorithm, and wrote the first commercial-grade software for solving LP problems [26]. His ideas have been employed and developed by, among others, Murtagh and M. A. Saunders [24], and Bixby [9, 6]. These efforts transformed the theory of linear programming from a theory into a powerful tool, and stimulated the swift development of the entire field.

On the other hand, Klee and Minty made explicit in 1972 that the simplex algorithm may require an exponential amount of work to solve LP problems [22]. At the early days of liner programming, Hoffman and Beal even revealed that cycling can occur with the simplex algorithm in the presence of degeneracy [20, 5]. Although anti-cycling techniques have been proposed from time to time, their practical performance turned out to be unsatisfactory [8, 12, 7, 27]. In practice, however, cycling rarely occurs despite frequent occurrence of degeneracy. Even so, many scholars believe that degeneracy degrades efficiency considerably, as the algorithm could stall at an degenerate vertex for too long a time before exiting it.

It seems that the impact of degeneracy is hard, if possible, to eliminate with the simplex algorithm, as it moves from vertex to vertex on the underlying polyhedron. In the past, some attempts were made to develop alternative algorithms by going

*Department of Mathematics, Southeast University, Nanjing, 210096, People's Republic of China (panpq@seu.edu.cn). Project 10871043 supported by National Natural Science Foundation of China.

away from this philosophy. The most important would be the development of the interior point algorithm, which approaches to an optimal solution by traveling across the interior of the polyhedron. Such kind of algorithms have achieved great success both theoretically and computationally.

The affine scaling algorithm, proposed by Dikin in 1967, is the first interior point algorithm for linear programming [13]. Many years later, Karmarkar proposed his famous polynomial-time algorithm in 1984, inspiring an upsurge of interior point algorithms [21]. This leads to some very efficient interior point algorithms for solving large sparse LP problems. However, these algorithms are not very suitable for cases where a vertex solution is required, as in a wide range of applications; in solution of integer and/or mixed integer linear programming problems, for instance, simples algorithms are still not replaceable at present [6].

**1.1. Main features of the proposed algorithm.** Then, it would be ideal for an algorithm to have advantages of both the simplex algorithm and the interior point algorithm while getting out of their disadvantages.

Along this line, Pan proposed algorithms that might better be termed *pivot algorithms* [28]–[35]. These algorithms allow a deficient basis, characterized as a submatrix from $A$ that has fewer columns than rows and whose range space includes $b$. Further, Pan proposed an algorithm that produces a sequence of interior points as well as a sequence of vertices, until reaching an optimal vertex [36]. Since it uses the QR decomposition to form projection on $A$'s null space, however, the algorithm may not be suitable for handling large scale sparse problems.

This paper is a continuation of the aforementioned endeavors. As described "nonstandard", the main features of the proposed algorithm are as follows:

1) It is based on the simplex framework: each iteration consists of pivoting operation and basis change, just as in the simplex algorithm.

2) It uses a new column rule.

3) It produces iterates that can be any feasible points. Starting with an interior point, in particular, it is just an interior point algorithm.

4) It can offer an approximate optimal vertex solution in certain sense.

5) It can achieve an optimal vertex solution if a simple purification is carried out.

The paper is organized as follows. Section 2 describes operations to determine a pivot and a search direction. Section 3 formulas the proposed nonstandard algorithm. Section 4 deals with the Phase-1 problem. Section 5 reports computational results, demonstrating the remarkable success of the nonstandard simplex algorithm to the standard simplex algorithm.

**2. Pivoting operations.** In the next section, we describe column and row pivoting operations. This topic is closely related to the determination of a search direction and the line search for updating iterates.

Let $B$ be the current basis and $N$ the associated nonbasis. Without confusion, denote *basic* and *nonbasic* index sets again by $B$ and $N$, respectively. For simplicity of exposition, we assume that $B$ is the submatrix consisting of the first $m$ columns: $A = [B\ N]$.

The reduced costs are defined by

$$\bar{c}_B = 0, \qquad \bar{c}_N = c_N - N^T\pi, \qquad B^T\pi = c_B. \tag{2.1}$$

Introduce nonbasic index sets

$$J_1 = \{j \in N \mid \bar{c}_j < -\varepsilon_1\}, \qquad J_2 = \{j \in N \mid \bar{c}_j > \varepsilon_1\}, \tag{2.2}$$

where $\varepsilon_1 > 0$ is a tolerance. In the following theoretical discussions, $\varepsilon_1 = 0$ is assumed.

Let $\bar{x}$ be the current feasible solution.

**2.1. Dantzig's column rule.** $\bar{x}$ is a basic feasible solution in the standard simplex context. So, values of its nonbasic components are equal to either of the associated lower and upper bounds. If the index set

$$\bar{J} = \{j \in J_1 \mid \bar{x}_j = l_j \} \cup \{j \in J_2 \mid \bar{x}_j = u_j \} \tag{2.3}$$

is empty, then an optimum is achieved, and all is done. In the other case, Dantzig's original rule determines an entering index $q$ such that

$$q = \text{argmax}\{ |\bar{c}_j| \mid j \in \bar{J} \}. \tag{2.4}$$

Consequently, the objective value will decrease the most (by amount $|\bar{c}_q|$) with the value of the associated variable increasing ($q \in J_1$) or decreasing ($q \in J_2$) by a unit.

A disadvantage of Rule (2.4) is that it does not take into account the amount by which the associated variable can increase or decrease. Even though $|\bar{c}_q|$ is large, the amount would be very small whenever so is $u_q - l_q$. In fact, the actual amount depends on the step-size that the variable takes on, which is subject to maintenance of feasibility; and this amount is too time consuming to compute.

**2.2. New column rule.** $\bar{x}$ is a feasible solution in our case. The new rule takes the ranges of the nonbasic variables into account. To this end, introduce notation

$$\delta_j = \begin{cases} u_j - \bar{x}_j & \text{if } j \in J_1, \\ \\ \bar{x}_j - l_j, & \text{if} j \in J_2, \end{cases} \tag{2.5}$$

Rule 1 (Column rule). *Select entering index $q$ such that*

$$|\bar{c}_q|\delta_q = max\{ |\bar{c}_j|\delta_j \mid j \in J_1 \cup J_2 \}. \tag{2.6}$$

In the standard simplex context, as a result, the objective value will decrease the most ( by amount $|\bar{c}_q|\delta_q$) if no broking basic variable exists for all nonbasic variables. However, no further progress can be made if $\delta_q = 0$.

Proposition 2.1 (Optimality condition). *$\bar{x}$ and $\bar{c}$ are a pair of primal and dual optimal basic solutions if it holds that*

$$\delta_j = 0, \qquad \forall j \in J_1 \cup J_2. \tag{2.7}$$

*Proof.* It is noted that $\bar{x}$ and $\bar{c}$ are primal and dual feasible solutions, respectively.

Condition (2.7) implies

$$\bar{c}_j\delta_j = 0, \qquad \forall j \in J_1 \cup J_2. \tag{2.8}$$

By (2.1) and (2.2), in addition, it holds that

$$\bar{c}_j = 0, \qquad \forall j \in B \cup N - (J_1 \cup J_2).$$

which along with (2.8) exhibits complementary slackness of $\bar{x}$ and $\bar{c}$. Therefore, they are a pair of primal and dual optimal basic solutions. □ It is now time to turn to

our general case when $\bar{x}$ is feasible but not necessarily basic. We will use Rule 1 to select an entering index.

In practice, what is required is often an approximate optimal solution only. We therefore modify condition (2.7) as follows.

DEFINITION 1. *$\bar{x}$ is an $\varepsilon$-optimal basic solution if*

$$\delta_j \le \varepsilon, \qquad j \in J_1 \cup J_2, \tag{2.9}$$

*where $\varepsilon > 0$ is a tolerance.*

**2.3. Search direction.** Assume now that condition (2.8) is not satisfied, and an entering index $q$ has been determined by Rule 1.

Define vector $\Delta x = (\Delta x_B^T \ \Delta x_N^T)^T$:

$$\Delta x_j = \begin{cases} -\text{sign}(\bar{c}_q), & j = q \\ \\ 0, & j \in N;\ j \ne q \end{cases} \tag{2.10}$$

$$\Delta x_B = \text{sign}(\bar{c}_q)\bar{a}_q, \qquad B\bar{a}_q = a_q. \tag{2.11}$$

Using the preceding notation, we have the following.

LEMMA 2.2. *The vector $\Delta x$ satisfies*

$$A\Delta x = 0, \qquad c^T\Delta x < 0. \tag{2.12}$$

*Proof.* From (2.10) and (2.11), it follows that

$$A\Delta x = \text{sign}(\bar{c}_q)BB^{-1}a_q - \text{sign}(\bar{c}_q)a_q = 0 \tag{2.13}$$

So, $\Delta x$ is in the null of $A$.

Further, by (2.1), it holds that

$$\bar{c}_q = c_q - c_B^T B^{-1} a_q, \tag{2.14}$$

which together with (2.10), (2.11), (2.6) and (2.2) gives

$$\begin{aligned} c^T\Delta x &= -\text{sign}(\bar{c}_q)c_q + \text{sign}(\bar{c}_q)c_B^T B^{-1}a_q \\ &= -\text{sign}(\bar{c}_q)(c_q - c_B^T B^{-1}a_q) \\ &= -\text{sign}(\bar{c}_q)\bar{c}_q \\ &< 0. \end{aligned} \tag{2.15}$$

□

Lemma 2.2 indicates the eligibility for $\Delta x$ being a descent search direction at $\bar{x}$, with respective to the objective.

**2.4. Row pivoting and determination of step-size.** Using $\Delta x$ as a search direction, we are led to the line search scheme $\hat{x} = \bar{x} + \alpha\Delta x$, or equivalently,

$$\hat{x}_B = \bar{x}_B + \alpha\Delta x_B, \tag{2.16}$$

$$\hat{x}_{k_q} = \bar{x}_q - \text{sign}(\bar{c}_q)\alpha, \tag{2.17}$$

$$\hat{x}_{k_j} = \bar{x}_j, \qquad j \in N;\ j \ne q. \tag{2.18}$$

where $\alpha$ is a step-length to be determined.

To make (2.16)–(2.18) an applicable update formula, the step-size $\alpha$ should be determined subject to $l \leq \hat{x} \leq u$. Using notation

$$\gamma_i = \begin{cases} (u_i - \bar{x}_i)/\Delta x_i & \text{if } \Delta x_i > 0, \text{ and } i \in B, \\ \\ (l_i - \bar{x}_i)/\Delta x_i & \text{if } \Delta x_i < 0, \text{ and } i \in B, \end{cases} \tag{2.19}$$

we derive the largest possible value of $\alpha$:

$$\bar{\alpha} = \min\{\delta_q, \min\{\gamma_i \mid \Delta x_i \neq 0, \ i \in B\}\}. \tag{2.20}$$

If $\bar{\alpha} = \delta_q$, there will be no basis change needed, (as will be handled later). If $\bar{\alpha} < \delta_q$, otherwise, the following rule is relevant.

Rule 2 (Row rule). *Select leaving index $p$ such that*

$$\bar{\alpha} = \gamma_p \quad \textit{for some} \quad p \in B \quad \textit{with} \quad \Delta x_p \neq 0 \tag{2.21}$$

Note: the preceding row rule is the same as the original one used in the simplex algorithm. It is not used in practice, however. Instead, usually used is Harries' more stable two-pass rule [19].

We will not take $\bar{\alpha}$ itself as the step-size but a smaller one instead:

$$\alpha = \lambda\bar{\alpha}, \qquad 0 < \lambda < 1. \tag{2.22}$$

Lemma 2.3. *$\hat{x}$ defined by (2.16)–(2.18) satisfies $A\hat{x} = b$ for any real $\alpha$.*

*Proof.* The validity follows from $A\bar{x} = b$, (2.16)–(2.18) and (2.12). □

A component of a feasible solution is said to be an *interior component* if its value is neither on the corresponding lower or upper bound; otherwise, it is said to be a *bound component*.

Although $\bar{x}$ is not a basic solution in general, it is still termed *nondegenerate* (associated with basis $B$) if its basic components are all interior, i.e.,

$$l_B < \bar{x}_B < u_B. \tag{2.23}$$

**3. The algorithm.** In this Section, we describe the proposed algorithm formally and make some theoretical analysis.

Assume that a new iterate $\hat{x}$ has been computed via (2.16)–(2.18 along with (2.20) and (2.23). If $\bar{\alpha}$ defined by (2.20) is equal to $\delta_q$, then no basis change is needed, and one can proceed on the next iteration. In the other case, a leaving index $p$ is determined by Rule 2; then, the next basis is formed from $B$ by replacing its $p$-th column with the $q$-indexed column of $A$. After the basis change, the next iteration is carried out.

The overall steps are the same as those in the standard simplex algorithm, except for pivoting operations. An efficient implementation involves maintenance of the LU decomposition of the basis. We only formula them in a conceptual manner as follows.

Algorithm 1. *Algorithm [Nonstandard simplex algorithm] Let $\bar{x}$ be the initial feasible solution and let $B$ be an initial basis. Given small numbers $\varepsilon > 0$ and $\varepsilon_1 > 0$. This algorithm solves (1.1).*

1. *Solve $B^T\pi = c_B$ for $\pi$.*
2. *Compute $\bar{c}_N = c_N - N^T\pi$.*

*3. Test for optimality:*
*(1) determine $J_1$ and $J_2$ by (2.2);*
*(2) compute $\delta_j, \ldots, j \in J_1 \cup J_2$ by (2.5);*
*(3) stop if it is satisfied that $\delta_j \leq \varepsilon$, $\forall j \in J_1 \cup J_2$.*
*4. Determine an entering index $q$ by Rule 1.*
*5. Solve $B\bar{a}_q = a_q$ for $\bar{a}_q$.*
*6. Compute $\Delta x$ by (2.10) and (2.11).*
*7. Compute $\bar{\alpha}$ by (2.20).*
*8. Stop if $\bar{\alpha}$ is too large.*
*9. Update $\bar{x}$ by (2.16)–(2.18 with (2.22).*
*10. Go to Step 1 if $\bar{\alpha} = \delta_q$.*
*11. Determine an leaving index $p$ by Rule 2.*
*12. Update basis $B$ by replacing its $p$-th column with $A$'s $q$-th column.*
*13. Go to Step 1*

Concerning the preceding algorithm, the following is clear.

THEOREM 1. *Assume termination of Algorithm 1. It terminates at either*
*(a) Step 3(3), achieving an $\varepsilon$-optimal basic solution; or*
*(b) Step 8, declaring lower unboundedness.*

At this stage, it has not been possible to rule out the possibility of infiniteness of Algorithm 1. Based on the fact that a large number of test problems have been solved with Algorithm 1 (see Section 5), however, we claim that Algorithm 1 must be regarded as finite practically, like the standard simplex algorithm.

Even if it is still open if Algorithm 1 is finite or not, in principle, we have the following results.

THEOREM 2. *Assume that the initial feasible solution is nondegenerate. Then all subsequent stepsizes are positive, and hence all iterates are nondegenerate feasible solutions. Moreover, the objective value strictly decreases.*

*Proof.* It is enough to consider a current iteration.

By lemma 2.3, the new iterate $\hat{x}$ satisfies $A\hat{x} = b$. From the feasibility of $\bar{x}$, (2.16)–(2.18), and (2.22), it follows that $\bar{x}$ satisfies $l \leq \hat{x} \leq u$, and hence is a feasible solution.

By (2.6), moreover, $\delta_q$ is positive, since, otherwise, the iteration would have terminated at Step 3(3). In addition, the nondegeneracy assumption implies that $\gamma_i$, defined by (2.19), are positive for all $i \in B$ satisfying $\Delta x_i \neq 0$. This fact along with (2.20) and (2.22) gives $0 < \alpha < \bar{\alpha}$. Consequently, from (2.16)–(2.18) it follows that $\hat{x}$ is again nondegenerate.

Furthermore, by $\alpha > 0$, (2.16)–(2.18) and Lemma 2.2) it holds that $c^T\hat{x} < c^T\bar{x}$. This completes the proof. □

Moreover, the algorithm has the following favorable features.

PROPOSITION 1. *Assume that the initial feasible solution is nondegenerate. A bound component of $\bar{x}$ could become an interior component; but any interior component never becomes a bound component.*

It is noted that *the algorithm proceed just like an interior point algorithm if the initial $\bar{x}$ is interior.* On the other hand, iterates generated are not vertices in general, even if the initial is. This is why Algorithm 1 bears the term "nonstandard".

**4. Phase-1 procedure.** Algorithm 1 needs a feasible solution and a basis to get itself started. We use a Phase-1 procedure for this purpose, which is similar to the typical one used in interior point approaches. We briefly describe it in this Section.

Any basis can be taken as an initial one although one that is "close" to an optimal is desirable. Assume now that a basis is available.

Assume that $x^0$ satisfies $l \le x^0 \le u$. Define $a_{n+1} = b - Ax^0$, and consider the auxiliary problem below:

$$\begin{array}{ll} \min & x_{n+1} \\ \text{s.t.} & Ax + x_{n+1}a_{n+1} = b, \qquad l \le x \le u,\ 0 \le x_{n+1} \le 1. \end{array} \tag{4.1}$$

The preceding has an optimal solution, as it has the obvious interior point $(x^0, 1)^T$, and its objective is bounded below.

Suppose that (4.1) has been solved by Algorithm 1 with the end solution $\bar{x}$. It is clear that the original program (1.1) has no feasible solution if the optimal value is greater than zero. In the other case, using $N$ and $B$ to denote the index sets such that $x_N^\infty = 0$ and $xbar_B > 0$, we recast program (1.1) into

$$\begin{array}{ll} \min & c_B^T x_B \\ \text{s.t.} & A_B x_B = b, \quad x_B \ge 0. \end{array} \tag{4.2}$$

Consequently, $barx_B > 0$ is available as an initial interior point to get the Phase-2 procedure started.

**4.1. Purification.** Algorithm 1 together with a simple purification can be used to obtain an exact optimal basic solution.

Assume that Algorithm 1 terminates at Step 3(3) with an $\varepsilon$-optimal basic solution. The purification is done by moving all nonbasic components of the solution onto their nearest bounds.

Let $x^0$ be the resulting solution from the purification. It is clearly an optimal basic solution if $Ax^0 = b$ is satisfied within a predetermined tolerance. In the other case, the standard two- phase simplex algorithm can be applied to obtain an optimal basic solution to the original problem, where the Phase-1 problem (4.2) is formed using $x^0$.

**5. Computational results.** Computational experiments have been performed to gain an insight into the behavior of Algorithm 3. We report numerical results in this section.

Implemented, and compared are the following three codes:

1) SSA: MINOS 5.51 with the full pricing option.

2) NSSA: Algorithm 1.

3) NSPR: Algorithm 1 with the purification.

In both NSSA and NSPR, $\varepsilon = 10^{-8}$ and $\varepsilon_1 = 10^{-8}$ were used for Phase-1, and $\varepsilon = 10^{-3}$ and $\varepsilon_1 = 10^{-6}$ were used for Phase-2.

Code NSSA and NSPR were developed using MINOS 5.51 as a platform. Therefore, the three codes shared such features as preprocessing, scaling, LUSOL [25], etc. Only the Mi50lp module was replaced by programs written by the author, with few other modules with minor changes. All codes used the default options, except for Rows 200000; Columns 300000; Elements 5000000; Iterations 20000000; Scale yes; Solution no; Log frequency 0; Print level 0.

Compiled using Visual Fortran 5.0, all codes were run under a Microsoft Windows XP Professional version 2002 on an ACER PC with an Intel(R) Pentium(R) 4 CPU 3.06GHz, 1.00GB memory, and about 16 digits of precision.

TABLE 5.1
*Statistics for 16 Kennington problems*

| Problem | SSA | | NSPR | | NSSA | |
|---|---|---|---|---|---|---|
| | Itns | Time | Itns | Time | Itns | Time |
| KEN-07 | 1724 | 1.97 | 1754 | 2.02 | 1750 | 2.02 |
| CRE-C | 4068 | 6.81 | 7487 | 11.89 | 2751 | 4.66 |
| CRE-A | 4066 | 7.67 | 8972 | 16.77 | 4069 | 8.09 |
| PDS-02 | 5110 | 9.06 | 1171 | 1.84 | 1025 | 1.61 |
| OSA-07 | 1767 | 7.16 | 1949 | 8.00 | 1204 | 5.02 |
| KEN-11 | 13327 | 111.39 | 13549 | 113.67 | 13542 | 113.59 |
| PDS-06 | 59521 | 433.67 | 8860 | 57.41 | 6317 | 40.36 |
| OSA-14 | 3821 | 33.48 | 3581 | 32.16 | 1747 | 16.19 |
| PDS-10 | 177976 | 2253.64 | 11191 | 122.13 | 7110 | 75.47 |
| KEN-13 | 35709 | 637.92 | 35301 | 629.66 | 35162 | 627.05 |
| CRE-D | 377243 | 4796.42 | 28042 | 358.75 | 1144 | 13.23 |
| CRE-B | 328160 | 4406.19 | 32228 | 430.81 | 1422 | 17.91 |
| OSA-30 | 7536 | 126.30 | 5886 | 101.59 | 2778 | 49.33 |
| PDS-20 | 1557953 | 44803.02 | 88068 | 2258.83 | 64777 | 1643.14 |
| OSA-60 | 16278 | 638.50 | 10976 | 449.45 | 4714 | 199.36 |
| KEN-18 | 157224 | 19395.88 | 155355 | 19315.88 | 154801 | 19253.55 |
| Total | 2751483 | 77669.08 | 414370 | 23910.86 | 304313 | 22070.58 |

Our first set of test problems included all 16 problems from Kennington[1]. The second set included all 17 problems from BPMPD[2] that were more than 500KB in compressed form.

In the tables, test problems are ordered by their sizes, in terms of m+n, where m and n are the numbers of rows and columns of the constraint matrix, excluding slack variables. All reported CPU times were measured in seconds with utility routine CPU_TIME, excluding the time spent on preprocessing and scaling. NSSA and NSPR used $10^{-6}$ as the equality tolerance (for holding of $Ax = b$), compared with SSA, which used $10^{-4}$, as usual.

**5.1. Results for Kennington test set.** Numerical results obtained with the 16 Kennington problems are displayed in Table 5.1, where the total iterations and time required for solving each problem are listed in the columns labeled Itns and Time under SSA, NSPR and NSSA, separately.

In Table 5.2, a performance comparison between the three codes is made by giving iteration and time ratios of SSA to NSPR, SSA to NSSA and NSPR to NSSA for each problem. It is seen from the bottom line that NSPR and NSSA outperformed SSA remarkably, with average iterations ratios 6.6 and 9.0, and time ratios 3.2 and 3.5! It is noted that the difference between NSPR and NSSA is small (with iterations ratio 1.4 and time ratio 1.1).

**5.2. Results for BPMPD test set.** Table 5.3 gives numerical results with the 17 BPMPD problems. Table 5.4 make a performance comparison between the codes by giving iteration and time ratios of SSA to NSPR, SSA to NSSA and NSPR to NSSA for each problem. It happened that SSA solved RADIO.PR and RADIO.DU in two or three iterations only. Excluding the meaningless results associated with them, therefore, we make the comparison only for the other 15 BPMPD problems.

It is seen from the second bottom line that NSPR and NSSA again outperformed SSA overall. The margins are large: the average iterations ratios are 2.8 and 9.5, and

[1] http://www-fp.mcs.anl.gov/otc/Guide/TestProblems/LPtest/
[2] http://www.sztaki.hu/~meszaros/bpmpd/

TABLE 5.2
*A comparison for 16 Kennington problems*

| Problem | M | N | SSA/NSPR | | SSA/NSSA | | NSPR/NSSA | |
|---|---|---|---|---|---|---|---|---|
| | | | Itns | Time | Itns | Time | Itns | Time |
| KEN-07 | 2427 | 3602 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| CRE-C | 3069 | 3678 | 0.5 | 0.6 | 1.5 | 1.5 | 2.7 | 2.6 |
| CRE-A | 3517 | 4067 | 0.5 | 0.5 | 1.0 | 0.9 | 2.2 | 2.1 |
| PDS-02 | 2954 | 7535 | 4.4 | 4.9 | 5.0 | 5.6 | 1.1 | 1.1 |
| OSA-07 | 1119 | 23949 | 0.9 | 0.9 | 1.5 | 1.4 | 1.6 | 1.6 |
| KEN-11 | 14695 | 21349 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| PDS-06 | 9882 | 28655 | 6.7 | 7.6 | 9.4 | 10.7 | 1.4 | 1.4 |
| OSA-14 | 2338 | 52460 | 1.1 | 1.0 | 2.2 | 2.1 | 2.0 | 2.0 |
| PDS-10 | 16559 | 48763 | 15.9 | 18.5 | 25.0 | 29.9 | 1.6 | 1.6 |
| KEN-13 | 28633 | 42659 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| CRE-D | 8927 | 69980 | 13.5 | 13.4 | 329.8 | 362.5 | 24.5 | 27.1 |
| CRE-B | 9649 | 72447 | 10.2 | 10.2 | 230.8 | 246.0 | 22.7 | 24.1 |
| OSA-30 | 4351 | 100024 | 1.3 | 1.2 | 2.7 | 2.6 | 2.1 | 2.1 |
| PDS-20 | 33875 | 105728 | 17.7 | 19.8 | 24.1 | 27.3 | 1.4 | 1.4 |
| OSA-60 | 10281 | 232966 | 1.5 | 1.4 | 3.5 | 3.2 | 2.3 | 2.3 |
| KEN-18 | 105128 | 154699 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Ave.(16) | - | - | 6.6 | 3.2 | 9.0 | 3.5 | 1.4 | 1.1 |

TABLE 5.3
*Statistics for 17 BPMPD problems*

| Problem | SSA | | NSPR | | NSSA | |
|---|---|---|---|---|---|---|
| | Itns | Time | Itns | Time | Itns | Time |
| RAT7A | 4073 | 573.59 | 4102 | 52.06 | 4100 | 52.03 |
| NSCT1 | 2190 | 35.83 | 13198 | 197.36 | 95 | 1.44 |
| NSCT2 | 11555 | 196.63 | 11971 | 182.23 | 92 | 1.45 |
| RIUTING | 190627 | 2143.70 | 75759 | 813.08 | 72774 | 780.00 |
| DBIR1 | 2289 | 47.44 | 12781 | 242.64 | 34 | 0.61 |
| DBIR2 | 42575 | 966.77 | 12750 | 235.83 | 12745 | 235.72 |
| T0331-4L | 62333 | 789.83 | 108780 | 1330.63 | 108575 | 1328.14 |
| NEMSEMM2 | 23581 | 204.59 | 18109 | 136.98 | 7327 | 58.00 |
| SOUTHERN | 29693 | 462.67 | 27550 | 314.17 | 27550 | 314.16 |
| WORLD.MO | 1405151 | 30953.42 | 822287 | 19193.38 | 94542 | 2929.00 |
| WORLD | 2627208 | 61340.78 | 1260355 | 27315.31 | 165045 | 4132.83 |
| NEMSEMM1 | 13650 | 263.20 | 14575 | 255.17 | 14491 | 253.64 |
| NW14 | 407 | 8.45 | 899 | 19.20 | 690 | 14.42 |
| LPL1 | 3343112 | 127379.39 | 203967 | 7214.02 | 2414 | 104.41 |
| DBIC1 | 624907 | 33335.58 | 377655 | 18746.17 | 375734 | 18679.69 |
| Total | 8383351 | 258701.88 | 2964738 | 76248.23 | 886208 | 28885.54 |
| RADIO.PR | 2 | 0.50 | 639702 | 14371.16 | 623810 | 14009.16 |
| RADIO.DU | 3 | 0.09 | 5055 | 56.11 | 5018 | 55.66 |

the time ratios are 3.4 and 9.0, respectively. On the other hand, the iterations ratio and time ratio of NSPR to NSSA are 3.3 and time ratio 2.6.

In the bottom line of Table 5.4, listed are average ratios for the entire set of the 31 test problems. The average iterations ratios of SSA to NSPR and of SSA to NSSA are 3.3 and 9.4, respectively. The associated time ratios are 3.4 and 6.6. So both NSPR and NSSA outperformed algorithm remarkable.

## REFERENCES

[1] I. Adler, N. Karmarkar, M. Resende and G. Veiga, Data structure and programming techniques for the implementation of Karmarkar's algorithm, *ORSA Journal on Computing*, **1**(1989),

TABLE 5.4
*A comparison for 15 BPMPD problems*

| Problem | M | N | SSA/NSPR | | SSA/NSSA | | NSPR/NSSA | |
|---|---|---|---|---|---|---|---|---|
| | | | Itns | Time | Itns | Time | Itns | Time |
| RAT7A | 3137 | 9408 | 1.0 | 11.0 | 1.0 | 11.0 | 1.0 | 1.0 |
| NSCT1 | 22902 | 14981 | 0.2 | 0.2 | 23.1 | 24.9 | 138.9 | 137.1 |
| NSCT2 | 23004 | 14981 | 1.0 | 1.1 | 125.6 | 135.6 | 130.1 | 125.7 |
| RIUTI | 20895 | 23923 | 2.5 | 2.6 | 2.6 | 2.7 | 1.0 | 1.0 |
| DBIR1 | 18805 | 27355 | 0.2 | 0.2 | 67.3 | 77.8 | 375.9 | 397.8 |
| DBIR2 | 18907 | 27355 | 3.3 | 4.1 | 3.3 | 4.1 | 1.0 | 1.0 |
| T0331 | 665 | 46915 | 0.6 | 0.6 | 0.6 | 0.6 | 1.0 | 1.0 |
| NEM.2 | 6944 | 42133 | 1.3 | 1.5 | 3.2 | 3.5 | 2.5 | 2.4 |
| SOUTH | 18739 | 35421 | 1.1 | 1.5 | 1.1 | 1.5 | 1.0 | 1.0 |
| WOR.M | 35665 | 31728 | 1.7 | 1.6 | 14.9 | 10.6 | 8.7 | 6.6 |
| WORLD | 35511 | 32734 | 2.1 | 2.2 | 15.9 | 14.8 | 7.6 | 6.6 |
| NEM.1 | 3946 | 71413 | 0.9 | 1.0 | 0.9 | 1.0 | 1.0 | 1.0 |
| NW14 | 74 | 123409 | 0.5 | 0.4 | 0.6 | 0.6 | 1.3 | 1.3 |
| LPL1 | 39952 | 125000 | 16.4 | 17.7 | 1384.9 | 1220.0 | 84.5 | 69.1 |
| DBIC1 | 43201 | 183235 | 1.7 | 1.8 | 1.7 | 1.8 | 1.0 | 1.0 |
| Ave.(15) | - | - | 2.8 | 3.4 | 9.5 | 9.0 | 3.3 | 2.6 |
| Ave.(31) | - | - | 3.3 | 3.4 | 9.4 | 6.6 | 2.8 | 2.0 |

No.2, 84–106.

[2] I. Adler, M. Resende, G. Veiga and N. Karmarkar, An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming*, **44**(1989), 297–335.

[3] E. R. Barnes, A variation of Karmarkars's algorithm for solving linear programming problems, *Mathematical Programming*, **36**(1986), 174–182.

[4] R. H. Bartels and G.H. Golub, The simplex method of linear programming using LU decomposition, *Communication ACM*, **12**(1969), 275–278.

[5] E. M. L. Beale, Cycling in the dual simplex algorithm, *Naval Research Logistics Quaterly*, **2**(1955), 269–275.

[6] R. E. Bixby, Solving real-world linear programs: A decade and more of progress, *Operations Research*, **50** (2002) No. 1, 3–15.

[7] R. G. Bland, New finite pivoting rules for the simplex method, *Mathematics of Operations Research* , **2**(1977),103–107.

[8] A. Charnes, Optimality and degeneracy in linear programming, *Econometrica*, **20** (1952), 160–170.

[9] ILOG CPLEX: High Performance Software of Mathematical Programming, http://www.ilog.com/products/cplex/.

[10] G.B. Dantzig, Programming in a linear structure, Comptroller, USAF, Washington, D.C. (February 1948).

[11] G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[12] 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** (1955), 183–195.

[13] I.I. Dikin, Iterative solution of problems of linear and quadratic programming, *Soviet Mathematics Doklady*, **8**(1967), 674–675.

[14] J.J.H. Forrest and D. Goldfarb, Steepest-edge simplex algorithms for linear programming, *Mathematical Programming* , **57** (1992), 341–374.

[15] J.J.H. Forrest and J.A. Tomlin, Updated triangular factors of the basis to maintain sparsity in the product form simplex method, *Mathematical Programming*, **2** (1972), 263–278.

[16] D.M. Gay, Electronic mail distribution of linear programming test problems, *Mathematical Programming Society COAL Newsletter* , **13**(1985), 10–12.

[17] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, A practical anticycling procedure for linearly constrained optimization, *Mathematical Programming*, **45** (1989), 437–474.

[18] H.-J. Greenberg, Pivot selection tactics, in H.J. Greenberg (Ed.) *Design and Implementation of Optimization software*, Sijthoff and Noordhoff, 1978, 109–143.

[19] P. M. J. Harris, Pivot selection methods of the Devex LP code, *Mathematical Programming Study*, **4** (1975), 30–57.

[20] A. J. Hoffman, Cycling in the simplex algorithm, Report No. 2974, Nat. Bur. Standards, Wash-

ington, D. C., 1953.
[21] N. Karmarkar, A new plolynomial time algorithm for linear programming, *Combinatorica* 4(1984), 373–395.
[22] V. Klee and G.J. Minty, How good is the simplex algorithm?, in:Inequalities, III(Edited by O. Shisha), Academic Press, New York, 1972, 159–175.
[23] K.O. Kortanek and M. Shi, Convergence results and numerical experiments on a linear programming hybrid algorithm, *European Journal of Operations Research*, **32**(1987), 47–61.
[24] B. A. Murtagh and M. A. Saunders, *MINOS 5.5 User's Guide*, Technical Report SOL 83-20R, Dept. of Operations Research, Stanford University, Stanford, 1998.
[25] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Maintaining LU factors of a general sparse matrix*, Linear Algebra Appl., 88/89 (1987), pp. 239–270.
[26] W. Orchard-Hays, Background development and extensions of the revised simplex method, Report RM 1433, The Rand Corporation, Santa Monica, California, 1954.
[27] P.-Q. Pan, Practical finite pivoting rules for the simplex method, *OR Spektrum*, **12** (1990), 219–225.
[28] P.-Q. Pan, The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study, *European J. of Operational Research*, **101** (1997), 167–176.
[29] P.-Q. Pan, A dual projective simplex method for linear programming, *Computers and Mathematics with Applications*, **35** (1998a), No. 6, 119–135.
[30] P.-Q. Pan, A basis-deficiency-allowing variation of the simplex method, *Computers and Mathematics with Applications*, **36** (1998b), No. 3, 33–53.
[31] P.-Q. Pan, A projective simplex method for linear programming, *Linear Algebra and Its Applications*, **292** (1999), 99–125.
[32] P.-Q. Pan, A projective simplex algorithm using LU factorization, *Computers and Mathematics with Applications*, **39** (2000), 187–208.
[33] P.-Q. Pan and Y. Pan, A phase-1 approach to the generalized simplex algorithm,*Computers and Mathematics with Applications*, **41** (2001), 1455–1464.
[34] P.-Q. Pan, A dual projective pivot algorithm for linear programming, *Computational Optimization and Applications*, **29** (2004), 333–344.
[35] P.-Q. Pan, A revised dual projective pivot algorithm for linear programming, *SIAM J. on Optimization*, **16**(2005), No.1, 49–68.
[36] P.-Q. Pan, An affine-scaling pivot algorithm for linear programming, submmitted.
[37] P.-Qi Pan, A largest-distance pivot rule for the simplex algorithm, *European Journal of Operations Research* , **187**(1998), No. 2, 393–402.
[38] P.-Q Pan, Efficient Nested Pricing in the Simplex Algorithm, *Operations Research Letters*, **38**(2008), 309-313.
[39] C. Roos, T.Terlaky, J.-Ph. Vial, Theory and algorithms for linear programming: An interior point approach, John Wiley, New York, 1997.
[40] T. Tsuchiya and M. Muramatsu, Global convergence of a long-step affine scaling algorithm for degenerate linear programming problems, *SIAM Journal on Optimization*,**5**(1995), No. 3, 525–551.
[41] T. Tsuchiya, Affine scaling algorithm, in *Interior Point Methods of Mathematical Programming*( T.Terlaky Ed.), Kluwer, 1996, 35-77.
[42] R.J. Vanderbei, M.S. Meketon and B.A. Freedman, A modification of Karmarkar's linear programming algorithm, *Algorithmica*, **1**(1986), 395–407.
[43] R.J. Vanderbei, and J.C. Lagarias, I.I. Dikin's convergence result for teh affine- scaling algorithm, *Contemporary Mathematics*, **114**(1990), 109–119.
[44] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwe Academic Publishers, Boston, 1996.
[45] Y.Ye, Interior point algorithms, John Wiley, New York, 1997.