Technical Memorandum Number 803

Improving the Efficiency of the Simplex Algorithm Based on a Geometric Explanation of Phase 1

by

Daniel Solow
Hans Halim

September 2005

Department of Operations
Weatherhead School of Management
Case Western Reserve University
330 Peter B Lewis Building
Cleveland, Ohio 44106

Improving the Efficiency of the Simplex Algorithm Based on a Geometric Explanation of Phase 1

Daniel Solow and Hans Halim

Department of Operations
Weatherhead School of Management
Case Western Reserve University
Cleveland, OH 44106
Fax: (216) 368–2650
e-mail: dxs8@po.cwru.edu

September 29, 2005

Abstract

An open question pertaining to the simplex algorithm is where phase 1 terminates on the feasible region. In this work, computer simulations support the conjecture that phase 1 terminates at a feasible point that is geometrically close, in terms of Euclidean distance, to the starting point, when compared to other feasible solutions. This observation is exploited to develop a coordinate transformation for bounded linear programs (LP) that typically results in phase 1 ending at a feasible solution that is geometrically close to the optimal solution, thus requiring fewer iterations in phase 2 to solve the problem. This is confirmed by extensive computational experiments using CPLEX on randomly generated LPs and also on the problems in Netlib and shows generally increasing benefits as the number of negative coefficients in the original objective function increases. Though not winning on all problems, the coordinate transformation saves an average of about 22% of the CPU time with CPLEX over all problems solved in Netlib.

1 Introduction and Notation

The question of where phase 1 terminates on the feasible region of a linear program (LP) is addressed in this paper. To that end, consider the following LP depicted in Figure 1.

Example 1: A Full-Dimensional Linear Program

$$
\begin{array}{lrcrcr}
\min & -x_1 & - & x_2 & & \\
\textbf{s.t.} & 2x_1 & + & 3x_2 & \leq & 26 \\
& 2x_1 & - & x_2 & \leq & 10 \\
& x_1 & - & 2x_2 & \leq & 2 \\
& -x_1 & + & 2x_2 & \leq & 8 \\
& x_1 & + & 3x_2 & \geq & 7 \\
& 3x_1 & - & x_2 & \geq & 1 \\
& 0 & \leq & x_1 & \leq & 8 \\
& 0 & \leq & x_2 & \leq & 7
\end{array}
$$

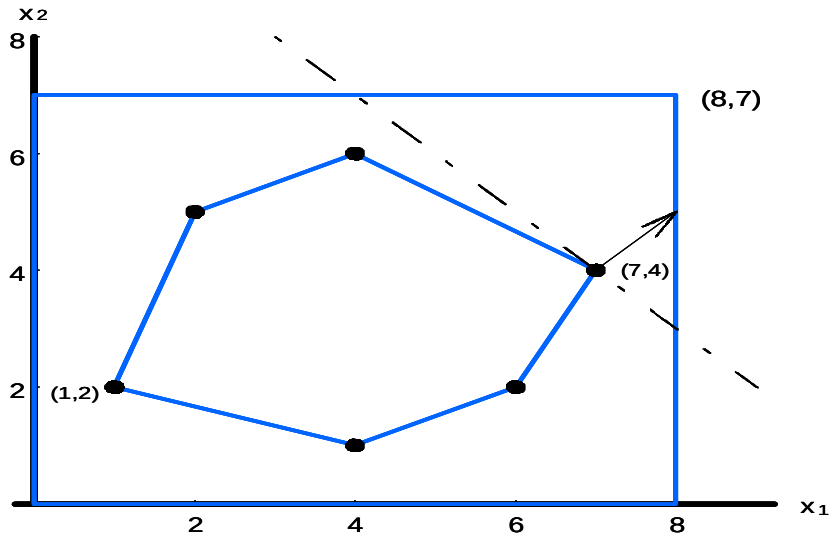

Figure 1: Geometry of Phase 1

When applying the simplex algorithm to solve the foregoing LP, an initial extreme point must first be found. The usual phase 1 procedure for doing so starts with $x_1 = x_2 = 0$ and ends with $x_1 = 1$ and $x_2 = 2$, in this case. As seen in Figure 1, this point is geometrically far, in Euclidean distance, from the optimal solution when compared to other extreme points and consequently one might expect that phase 2 requires many iterations before obtaining a solution. Numerous researchers have attempted to remedy this problem by developing algorithms that travel through the interior of the feasible region and it was not until the work of Karmarkar (1984) that a computationally efficient method was found, at least for certain large LPs.

In this paper, rather than trying to go through the interior of the feasible region, a coordinate transformation is used in an attempt to arrange for phase 1 to terminate at an initial extreme point on the "proper" side of the feasible region, that is, geometrically close to the optimal solution. (From here on, "distance" refers to Euclidean distance.) To illustrate with the LP in Example 1, turn Figure 1 upside down and imagine that the point $x_1 = 8$ and $x_2 = 7$ is the origin of a new w-coordinate system. Phase 1, when applied to the transformed problem, terminates with the optimal solution to the original LP and hence phase 2 is not required.

The choice of coordinate system is based on several conjectures which are believed to hold generally, although it is possible to produce specific counterexamples. The first one is the following.

Conjecture 1: The farther a basic feasible solution (bfs) is from the optimal solution when compared to other bfs's, the more iterations are required in phase 2.

If Conjecture 1 is true, then it might be computationally advantageous to arrange for phase 1 to terminate at a bfs that is geometrically close to the optimal solution so as to reduce iterations in phase 2. A coordinate transformation that accomplishes this is proposed in Section 2. The coordinate transformation is based on the following conjecture regarding where phase 1 terminates.

Conjecture 2: Phase 1 terminates at a feasible point that is geometrically close to the starting point—that is, to the initial values of the variables—when compared to other feasible points.

If Conjectures 1 and 2 are generally true, then it is desirable to identify a starting point for phase 1 that is geometrically close to the optimal solution when compared to other feasible points. Assuming that phase 1 starts at the origin, an equivalent objective is to identify a coordinate transformation so the origin of this new w-coordinate system is geometrically close to the optimal solution (which remains invariant under the transformation) when compared to other feasible points. Such a coordinate transformation for bounded LPs is proposed in Section 2. The result of the transformation is an equivalent LP that can be solved using any state-of-the-art simplex code, such as CPLEX. Results of computer simulations on randomly-generated LPs are presented in Section 2 to justify Conjectures 1 and 2 and show the computational advantage of solving the recommended transformed problem. On these problems, savings in the phase 2 iterations of the transformed problems are shown to increase from 0% up to 50%, as the fraction of negative components in the objective function increases from 0 up to 100%, while the number of phase 1 iterations remains about the same.

In Section 3, several implementation issues associated with the coordinate transformation are discussed and the proposed solutions are supported by computer simulations. The results of applying the coordinate transformation to the real-world test problems in Netlib are reported in Section 4 and again support Conjectures 1 and 2. Consistent with the randomly-generated problems, these results show a generally increasing reduction in the phase 2 iterations of the transformed problem over the original problem as the fraction of negative components in the original objective function increases. Those results also indicate that, although the transformed problems generally require more phase 1 and total iterations than the original problem, an overall average savings of about 22% in CPU time is realized by solving the transformed Netlib problems.

2 Coordinate Transformations for Bounded Linear Programs

For bounded LPs, a coordinate transformation is now proposed with the property that the origin of the transformed problem is geometrically close to the optimal solution compared to other feasible points. To that end, let A be an $(m \times n)$ matrix, $\mathbf{c}, \mathbf{l}, \mathbf{u} \in R^n$, and $\mathbf{b} \in R^m$, and assume that the original LP is in the following form:

$$\begin{array}{ll} \min & \mathbf{cx} \\ \text{s.t.} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{array} \tag{1}$$

Throughout, let A_j denote column j of A.

2.1 The Transformed Linear Program

Each of the 2^n extreme points of the n-dimensional box formed by the lower and upper bounds of the LP in (1) is a potential origin for the new w-coordinate system (see Figure 1). The first step is to identify, with minimal computational effort, an extreme point of this box that is close to the (unknown) optimal solution to (1) when compared to other feasible points. To that end, temporarily ignore the constraints $A\mathbf{x} = \mathbf{b}$ in (1). By inspection, the optimal solution to the

resulting LP is:

$$x_j^* = \begin{cases} u_j, & \text{if } c_j < 0 \\ l_j, & \text{if } c_j \geq 0 \end{cases} \tag{2}$$

Thus, the third conjecture is the following:

Conjecture 3: The point in (2) obtained by dropping the equality constraints from the original LP is geometrically close to the optimal solution of the LP in (1), when compared to other feasible points.

Based on Conjecture 3, the following transformation is now proposed, in which the origin of the new w-coordinate system is given in (2):

$$x_j = \begin{cases} u_j - w_j, & \text{if } c_j < 0 \\ l_j + w_j, & \text{if } c_j \geq 0 \end{cases} \tag{3}$$

Substituting the values for the variables x_j from (3) in the LP in (1) results in the following equivalent **transformed LP**:

$$\begin{array}{ll} \min & \bar{\mathbf{c}}\mathbf{w} + \bar{\bar{\mathbf{c}}} \\ \text{s.t.} & \bar{A}\mathbf{w} = \bar{\mathbf{b}} \\ & \mathbf{0} \leq \mathbf{w} \leq \bar{\mathbf{u}} \end{array} \tag{4}$$

in which $\bar{\mathbf{c}} \geq \mathbf{0}$ and $\bar{\mathbf{c}}$, $\bar{\bar{\mathbf{c}}}$, $\bar{A}$, $\bar{\mathbf{b}}$, and $\bar{\mathbf{u}}$ are defined as follows:

$$\bar{c}_j = \begin{cases} -c_j, & \text{if } c_j < 0 \\ c_j, & \text{if } c_j \geq 0 \end{cases} \qquad \bar{\bar{\mathbf{c}}} = \sum_{j=1}^{n} c_j \cdot \begin{cases} u_j, & \text{if } c_j < 0 \\ l_j, & \text{if } c_j \geq 0 \end{cases} \qquad \bar{\mathbf{u}} = \mathbf{u} - \mathbf{l}.$$

$$\bar{A}_j = \begin{cases} -A_j, & \text{if } c_j < 0 \\ A_j, & \text{if } c_j \geq 0 \end{cases} \qquad \bar{\mathbf{b}} = \mathbf{b} - \sum_{j=1}^{n} A_j \cdot \begin{cases} u_j, & \text{if } c_j < 0 \\ l_j, & \text{if } c_j \geq 0 \end{cases}$$

Because $\bar{\mathbf{c}} \geq \mathbf{0}$, one might expect that the point $\mathbf{w} = \mathbf{0}$ is a good starting point for phase 1 because $\mathbf{w} = \mathbf{0}$ provides the smallest possible objective function value for (4).

One desirable feature of the coordinate transformation is the ease of computational implementation. The data $\bar{A}$, $\bar{\mathbf{c}}$, $\bar{\bar{\mathbf{c}}}$, $\bar{\mathbf{b}}$, and $\bar{\mathbf{u}}$ for the transformed LP in (4) are obtained by preprocessing the data of the original LP in (1). Once this is done, any state-of-the-art simplex code can be used to solve (4). One need only make sure that its phase 1 procedure is initiated with $\mathbf{w} = \mathbf{0}$. Care is needed if the code employs a crashing technique [Gould and Reid (1989) and Carstens (1968)] or other recent advances in which phase 1 is not initiated at the origin [see, for example, Bixby (1992) and Maros (1986)]. On obtaining an optimal solution $\mathbf{w}^*$ to (4), a simple inverse transformation produces an optimal solution $\mathbf{x}^*$ to (1).

The result is that the transformed LP in (4) is equivalent to the LP in (1), as stated in the following theorem, whose proof is clear from the foregoing transformation formulas.

Theorem 2.1 *The transformed LP in (4) is infeasible or optimal if and only if the original LP in (1) is infeasible or optimal, respectively. Furthermore, if* $\mathbf{w}^*$ *is optimal for the LP in (4), then* $\mathbf{x}^*$ *as defined in (3) is optimal for the LP in (1).*

2.2 Computer Simulation Results Justifying the Coordinate Transformation

The coordinate transformation suggested in Section 2.1 is predicated on Conjectures 1, 2, and 3. Although no proof for these conjecture has been found, convincing evidence comes from examining the number of iterations needed to solve randomly-generated problems. If these conjectures are correct, then one would expect that the farther the optimal solution is from the origin of an LP relative to other basic feasible solutions, the more work it should take in phase 2 to solve the problem.

To test this hypothesis, it is necessary to generate LPs in which the "distance" from the optimal solution to the origin can be controlled. From the solution in (2), one measure of this distance is the number of negative objective function coefficients in the original LP. That is, if none of the components of the vector $\mathbf{c}$ is negative in the original LP, then the optimal solution is expected to be close to the origin and hence should require few iterations in phase 2. At the other extreme, if all components of $\mathbf{c}$ are negative, then the optimal solution is expected to be far from the origin and thus should require more iterations in phase 2 than when the optimal solution is close to the origin. In summary, if these conjectures are correct, then the more the number of negative components of $\mathbf{c}$ in randomly-generated LPs, the greater the number of iterations that should be required in phase 2 to obtain the optimal solution.

The results of such a simulation are presented now. Each replication of the simulation has the following steps:

1. Create a random LP as follows:
   (a) Define the size of the LP (m is number of constraints and n is number of variables) and define p as the fraction of negative components of the vector $\mathbf{c}$.
   (b) Define the vector $\mathbf{c}$ by generating n random numbers between 0 and 1 from a uniform distribution.
   (c) Let $k = \lfloor pn \rfloor$ and multiply the first k components of the vector $\mathbf{c}$ by -1.
   (d) For each element of the matrix A, generate a random number between -1 and 1 from a uniform distribution.
   (e) Assign each component of the lower bound $\mathbf{l}$ a value of 0.
   (f) Define the upper bounds $\mathbf{u}$ by generating n random numbers between 0 and 1 from a uniform distribution.
   (g) To ensure a feasible LP, generate n random numbers between 0 and u_i from a uniform distribution, calling the resulting vector $\mathbf{z}$. Then set $\mathbf{b} = A\mathbf{z}$.
2. Solve the LP created in Step 1.
3. Apply the coordinate transformation described in Section 2.1 and solve the transformed LP.

CPLEX (version 8.1) was used to solve the resulting LPs. All computations were performed on a Pentium IV 2.80 GHz machine with 1 GBytes RAM using Windows XP as the operating system. The randomly generated problems have 100 constraints and 200 variables. Two hundred problems were generated for each value of p, starting with no negative components of the vector $\mathbf{c}$ ($p = 0.0$) and ending with all negative components of the vector $\mathbf{c}$ ($p = 1.0$) by increments of 0.1.

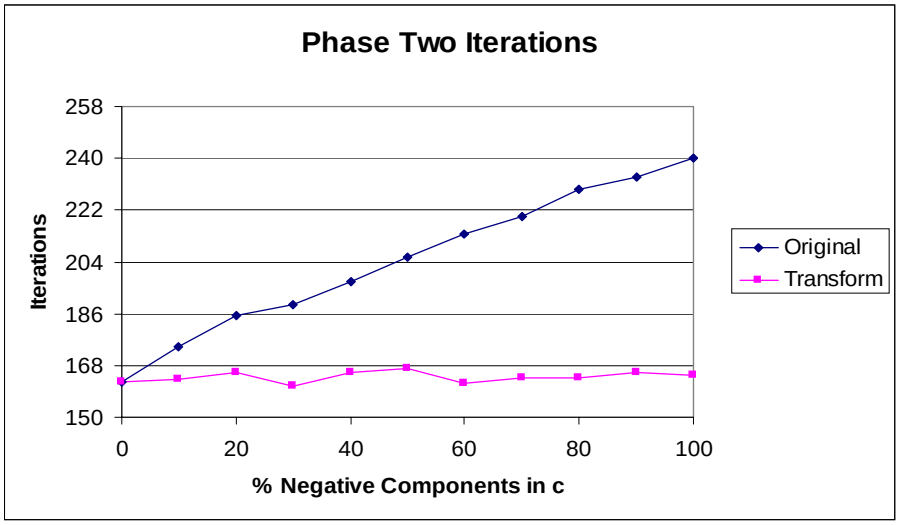

Figure 2: Effect of Percent of Negative Components of **c** on Iterations in Phase 2

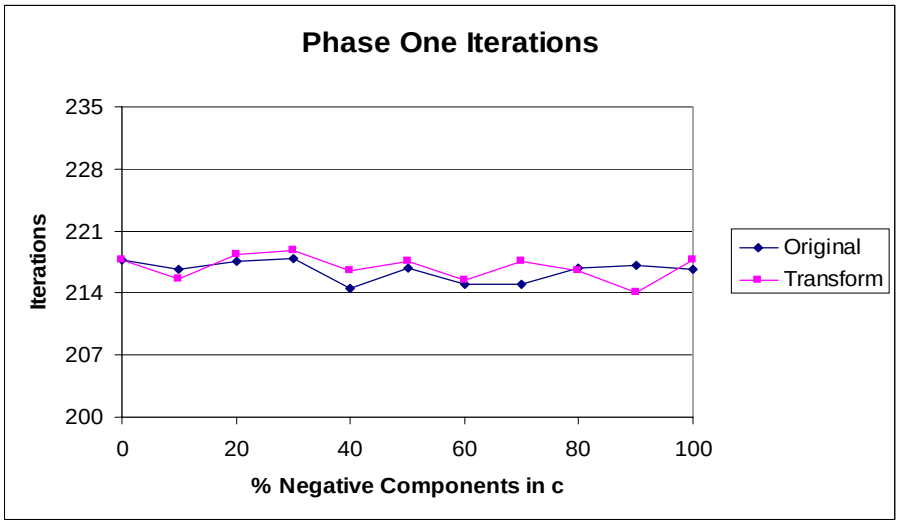

Figure 3: Effect of Percent of Negative Components of **c** on Iterations in Phase 1

The x-axis in Figure 2 represents the percent of negative components in the vector **c** and the y-axis represents the average number of iterations in phase 2 needed to obtain the optimal solution. Similar information pertaining to the phase 1 iterations and the total number of iterations needed to solve the problems are shown in Figure 3 and Figure 4.

The upper curves in Figure 2 and Figure 4 correspond to the original LP and the lower curves to the transformed LP. As can be seen, the more the number of negative components of **c**, the greater the average number of iterations needed in phase 2 (and hence in total) to solve the original problem. This confirms Conjectures 1, 2, and 3 in that the more the number of negative components of **c**, the farther the optimal solution is expected to be from the origin and hence the more the number of iterations should be required in phase 2 to solve the original problem.

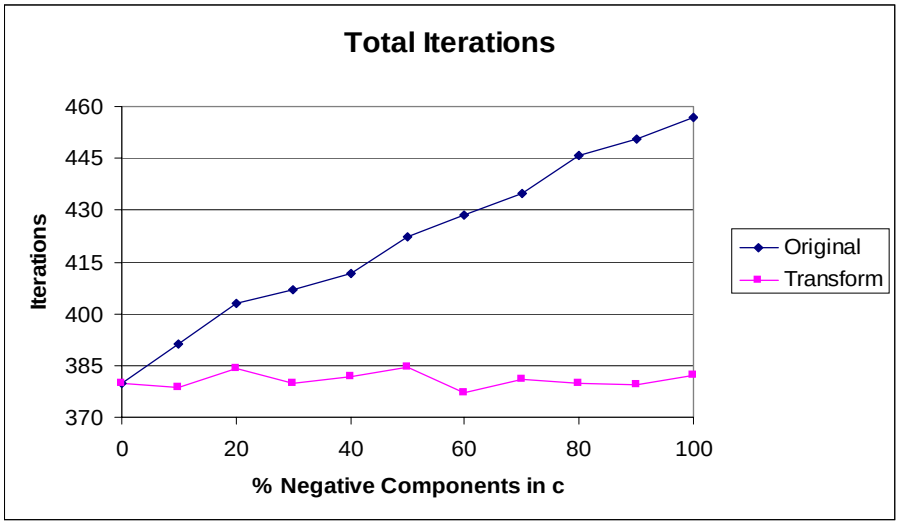

Figure 4: Effect of Percent of Negative Components of **c** on Iterations in Total

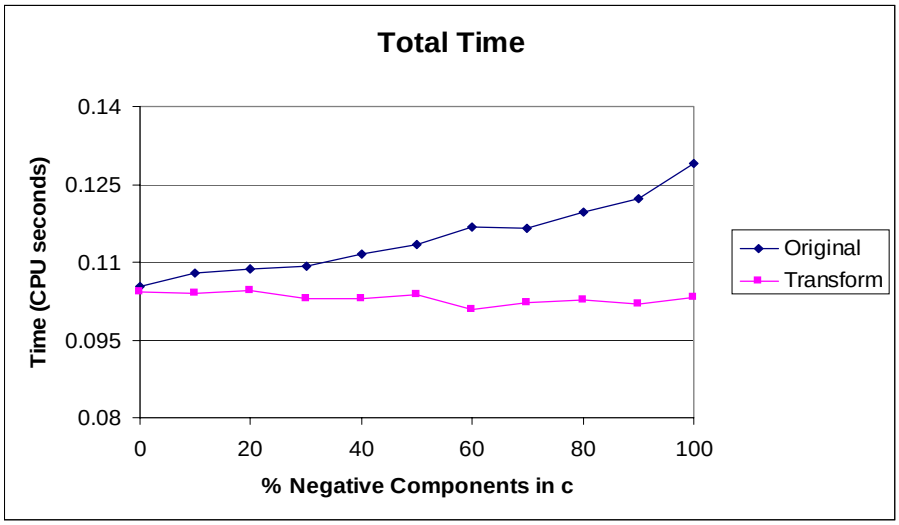

Figure 5: Effect of Percent of Negative Components of **c** on Total Time

The lower curve in Figure 4 indicates that the transformed problem requires fewer iterations on average than the original problem, with the benefits increasing as the number of negative components of **c** increases. Furthermore, the two curves in Figure 3 indicate that the number of iterations in phase 1 for the transformed LP and the original LP are relatively independent of the number of negative components in **c**. This is because random LPs are "centered" within the lower and upper bounds, and so the distance from any extreme point of the bounds to the feasible region is approximately the same. Also notice that the total number of iterations needed to solve the transformed problem is relatively independent of the number of negative components in **c**. This is because, with randomly-generated problems, the optimal solution is always in approximately the same position relative to the origin of the transformed coordinate system.

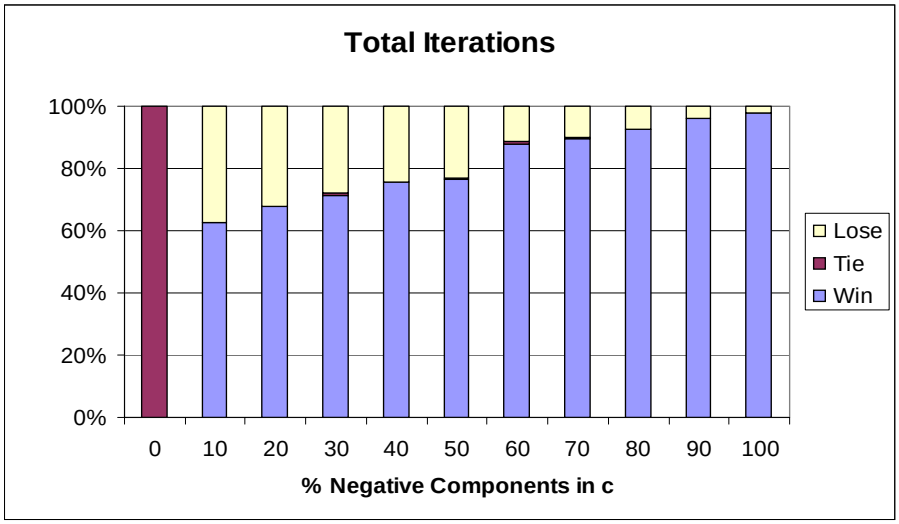

Figure 6: Effect of Percent of Negative Components of **c** on the % of Wins/Losses/Ties in Total

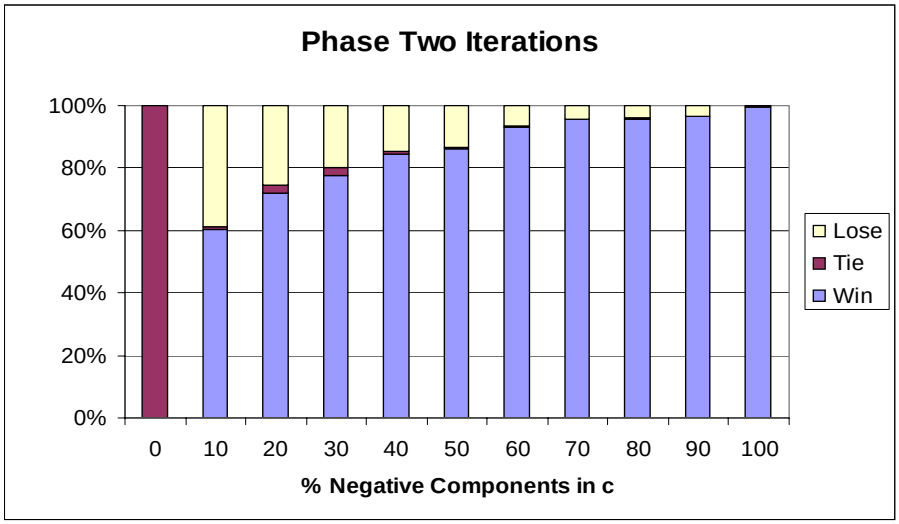

Figure 7:Effect of Percent of Negative Components of **c** on the % of Wins/Losses/Ties in Phase 2

The x-axis in Figure 5 represents the percent of negative components in the vector **c** and the y-axis represents the average total time needed to obtain the optimal solution. The times reported are those returned by the "time" C-library function. The upper curve in Figure 5 corresponds to the original LP and the lower curve to the transformed LP. As can be seen, the more the number of negative components of **c**, the more time needed to solve the original problem. Therefore, there is a positive correlation between the number of iterations needed to obtain the optimal solution and the total time needed to obtain the optimal solution for randomly-generated LPs.

It is not possible to conclude from Figures 2, 4 and 5 that the coordinate transformation will always be more efficient as the number of negative components of **c** increases. This is illustrated in Figure 6, where the x-axis represents the percent of negative components in the vector **c** and

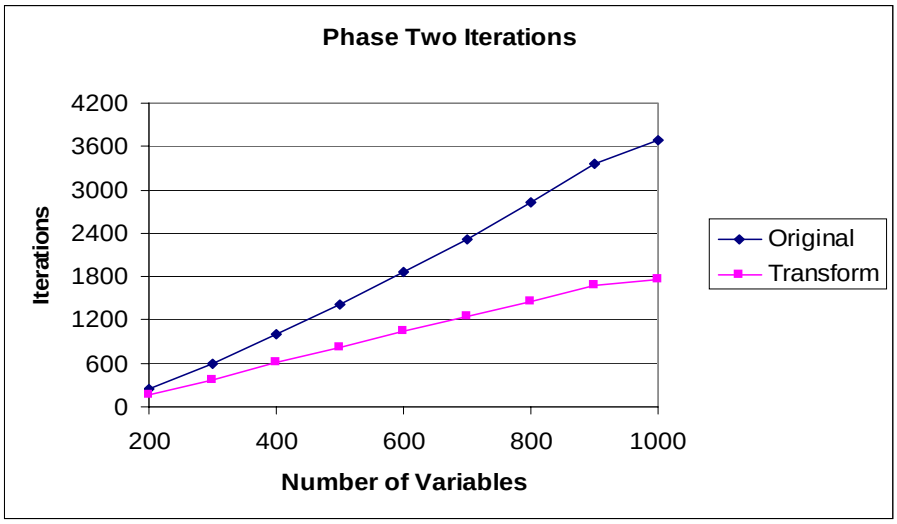

Figure 8: Effect of Number of Variables on Iterations in Phase 2

the y-axis represents the percent of wins, losses and ties in the total number of iterations to obtain the optimal solution for the transformed problems compared to the original problems. Similar information pertaining to phase 2 is shown in Figure 7. As can be seen, the more the number of negative components of $\mathbf{c}$, the higher the percent of wins on average in phase 2 for the transformed problems compared to the original problems. Since the average number of iterations in phase 1 for the original LP and the transformed LP are approximately the same, the more the number of negative components of $\mathbf{c}$, the more the number of wins on average in total for the transformed problems compared to the original problems. In summary, the coordinate transformation is likely—but not guaranteed—to win as the number of negative components of $\mathbf{c}$ increases.

2.3 Sensitivity Analysis with Regard to Problem Size

An important question is how sensitive the benefits of the transformed problem are to the size of the LP in terms of the number of constraints and the number of variables. Therefore, a simulation was conducted by fixing the number of constraints to 100, assigning all negative components to $\mathbf{c}$, and increasing the number of variables. The results of solving 200 randomly generated problems of each size are shown in Figure 8 and Figure 9. The x-axis in Figure 8 represents the number of variables and the y-axis represents the average number of iterations in phase 2 needed to obtain the optimal solution. Similar information pertaining to the total number of iterations needed to solve the problem is shown in Figure 9.

The upper curves in Figure 8 and Figure 9 correspond to the original LP and the lower curves to the transformed LP. As expected, the more the number of variables, the greater the average number of iterations needed in phase 2 (and hence in total) to solve the original problem and the transformed problem. However, the average number of iterations in phase 2 and in total grows at a slower rate for the transformed problem.

Another simulation was conducted by fixing the number of variables to 500, assigning all negative components to $\mathbf{c}$, and increasing the number of constraints from 50 to 450. The results of these simulations are shown in Figure 10 and Figure 11. The x-axis in Figure 10 represents the

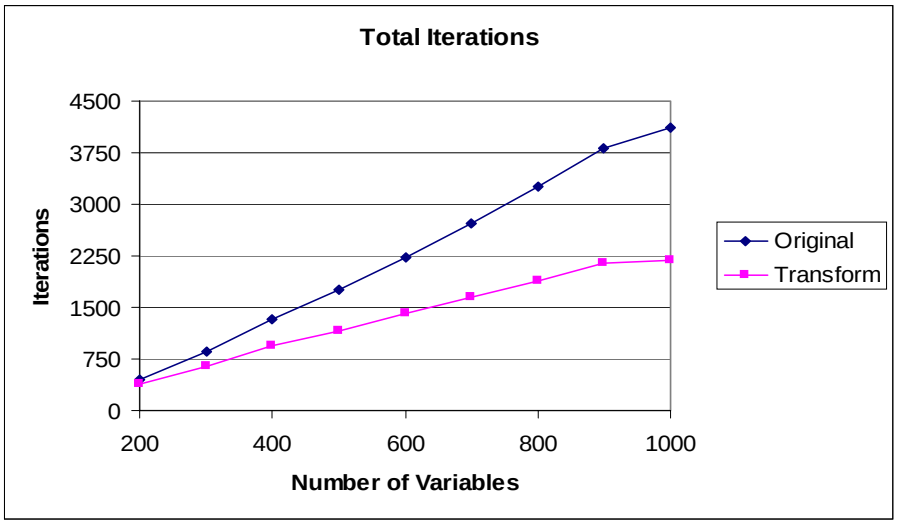

Figure 9: Effect of Number of Variables on Iterations in Total

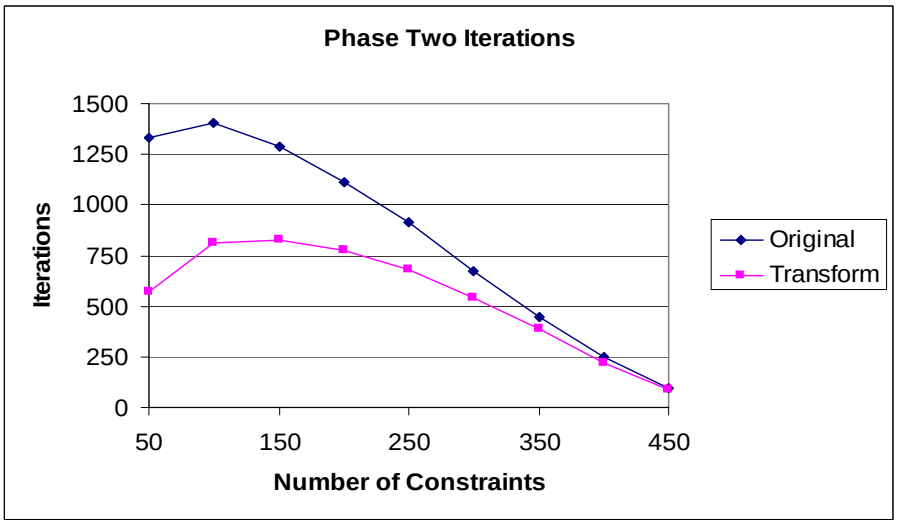

Figure 10: Effect of Number of Constraints on Phase 2 Iterations

number of constraints and the y-axis represents the average number of iterations in phase 2 needed to obtain the optimal solutions. Similar information pertaining to the average total number of iterations needed to solve the problems is shown in Figure 11.

The upper curves in Figure 10 and Figure 11 correspond to the original LP and the lower curves to the transformed LP. As can be seen, the transformed LP takes fewer iterations on average, regardless of the number of constraints. Also, the more the number of constraints, the fewer the average number of iterations needed in phase 2 (and hence in total) to solve the original problem and the transformed problem, after a certain number of constraints (100 for the original LP and 150 for the transformed LP). The decrease in iterations is because, as the number of constraints increases to the number of variables, the matrix A in the equality constraints will become square.

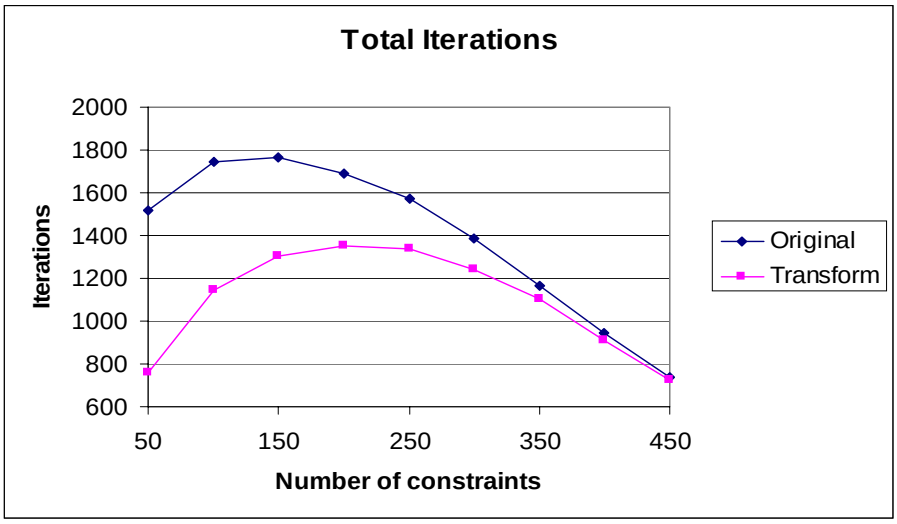

Figure 11: Effect of Number of Constraints on Total Iterations

A final observation is that the more the number of constraints, the smaller the savings in average number of iterations in phase 2 and in total.

In summary, the transformed problem benefits regardless of the size of the LP. Moreover, the greater the difference between the number of variables and the number of constraints, the greater the benefits the transformed problem can achieve.

3 Computational Implementation Issues

The simulations presented in Section 2.2 show the potential computational savings that can be achieved when solving the transformed LP. To realize those savings on real-world problems, a number of implementation issues are addressed in this section.

3.1 Sensitivity to the Bounds

On randomly generated LPs whose feasible regions are "centered" within their bounds, one would expect the number of phase 1 iterations to be independent of the coordinate system. This is the case with the random problems in the simulations in Section 2.2. However, if the feasible region is far from the bounds used for performing the transformation in (3), then the benefits gained in phase 2 from the transformed problem could be offset by the additional number of iterations required in phase 1.

A simulation was therefore performed to determine how sensitive the benefits of the transformed problem are to the size of the bounds determining the transformation. Random problems with all negative components in the $\mathbf{c}$ vector, 100 constraints and 200 variables are generated. Then, the original problem and transformed problem are solved many times, each time the original upper bounds are multiplied by $1.1, 1.2, \ldots, 1.9, 2, 4$.

The results of these simulations are shown in Figures 12, 13, and 14. The x-axis in Figure 12 represents the value of the upper-bound multipliers and the y-axis represents the average number

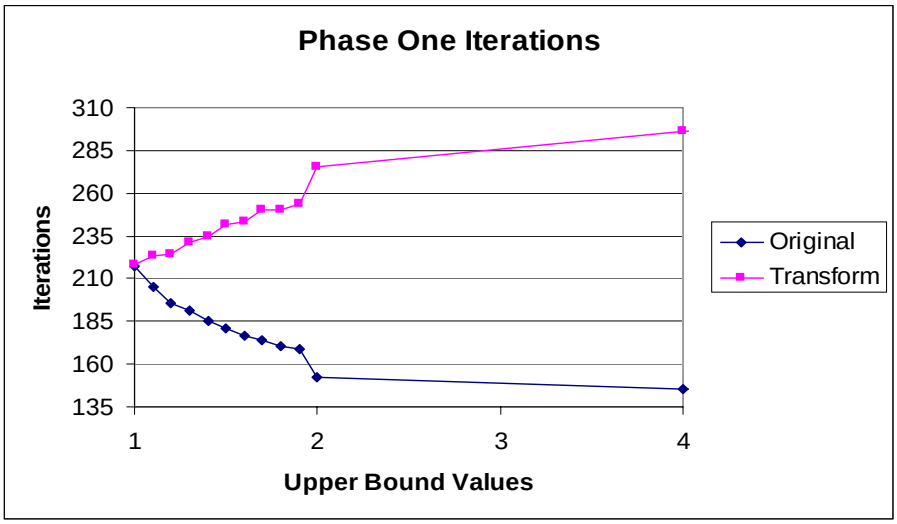

Figure 12: Effect of Upper Bounds on Phase 1 Iterations

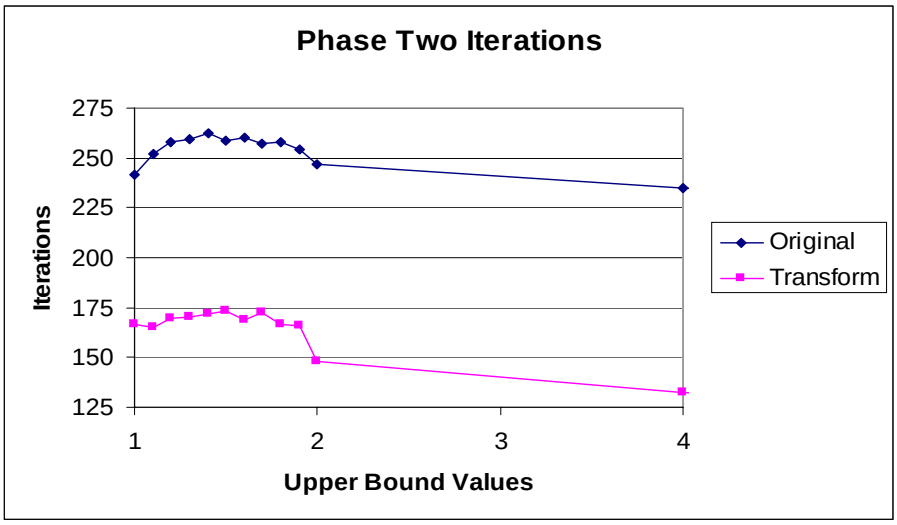

Figure 13: Effect of Upper Bounds on Phase 2 Iterations

of iterations in phase 1. Similar information pertaining to the average number of iterations in phase 2 and total number of iterations is shown in Figure 13 and Figure 14. The upper curve in Figure 12 shows that as the bounds of the original LP that determine the transformation increase—thus meaning that the starting point of the transformed problem gets farther from the feasible region—the number of phase 1 iterations needed quickly increases. As seen in Figure 14, with this increase in phase 1 iterations, the total number of iterations needed to solve the transformed LP increases as the determining bounds increase. When original upper bounds are multiplied by 2.0, the transformed LP is not performing as well as the original LP. Therefore, the conclusion is that tighter bounds benefit the transformed problem.

It is also interesting to note from the lower curve in Figure 12 that, as the determining bounds

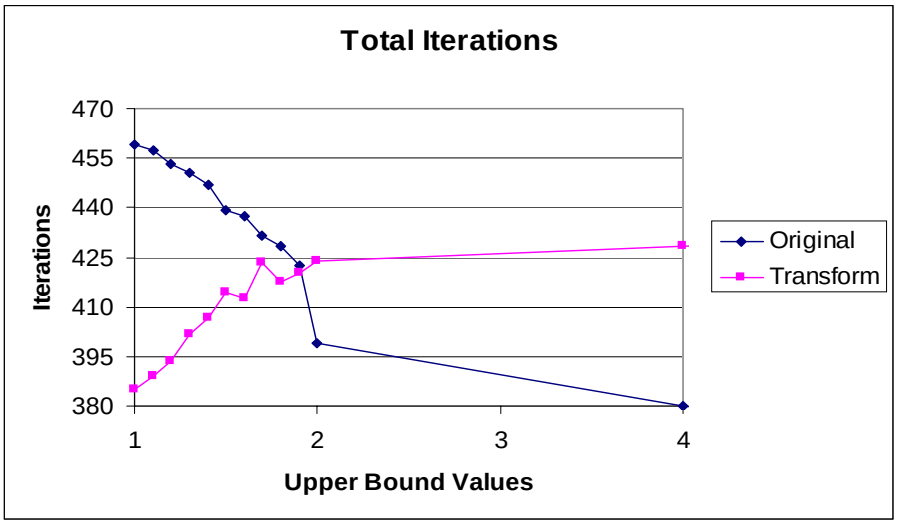

Figure 14: Effect of Upper Bounds on Total Iterations

increase, the number of phase 1 iterations needed to solve the original problem actually decreases. This is because, as these bounds increase, the values of the basic variables in the starting solution of phase 1, which remains constant in the original problem as the bounds increase, satisfy an increasing number of the bounds constraints, thus requiring less work in phase 1 to find a feasible solution.

3.2 CPLEX Issues

In this section, it is shown that the current version of CPLEX is sensitive to a number of factors, including something as simple as a shift in origin. To illustrate, consider an original LP in form (1). If the origin of the coordinate system is shifted by letting $\mathbf{x} = \mathbf{w} + \mathbf{l}$, then the shifted LP takes the following form:

$$\begin{array}{ll} \min & \mathbf{cw} + \mathbf{cl} \\ \text{s.t.} & A\mathbf{w} = \mathbf{b} - A\mathbf{l} \\ & \mathbf{0} \leq \mathbf{w} \leq \mathbf{u} - \mathbf{l} \end{array} \tag{5}$$

Mathematically and geometrically, (5) is equivalent to the original problem. Therefore, one would expect that an LP in form (1) or form (5) would require the same number of phase 1 and phase 2 iterations. However, this is not the case with CPLEX.

To show this anomaly, two hundreds random LPs were generated in which all lower bounds are negative and all upper bounds are positive. After solving these LPs, their origins are shifted so that all lower bounds are zero. In this simulation, the number of constraints is fixed to 100, the number of variables to 200, and the percent of negative components of the vector $\mathbf{c}$ is increased gradually.

The x-axis in Figure 15 represents the percent of negative components in the vector $\mathbf{c}$ and the y-axis represents the number of iterations in phase 2 needed to obtain the optimal solution. Similar information pertaining to the total number of iterations is shown in Figure 16.

As can be seen, the more the number of negative components of $\mathbf{c}$, the fewer the average number of iterations needed in phase 2 (and hence in total) to solve the original problem. In contrast, the

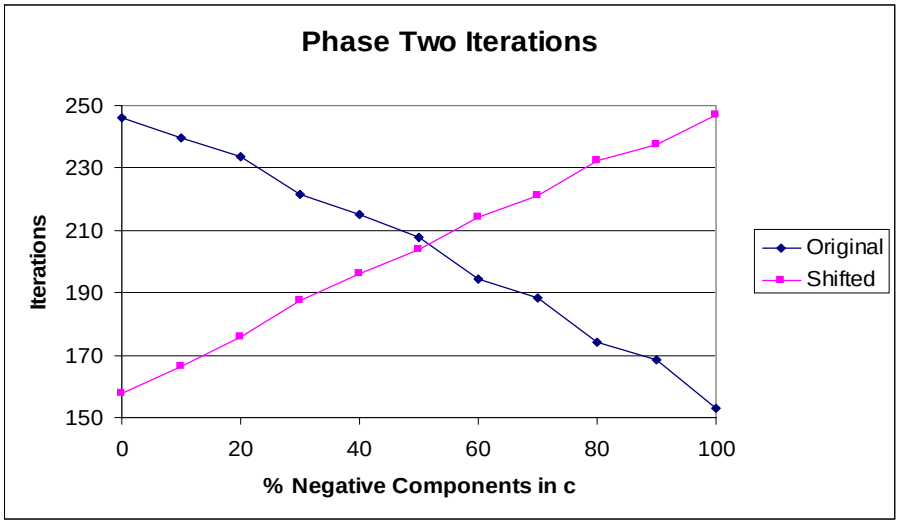

Figure 15: Effect of Percent of Negative Components of **c** on Phase 2 Iterations

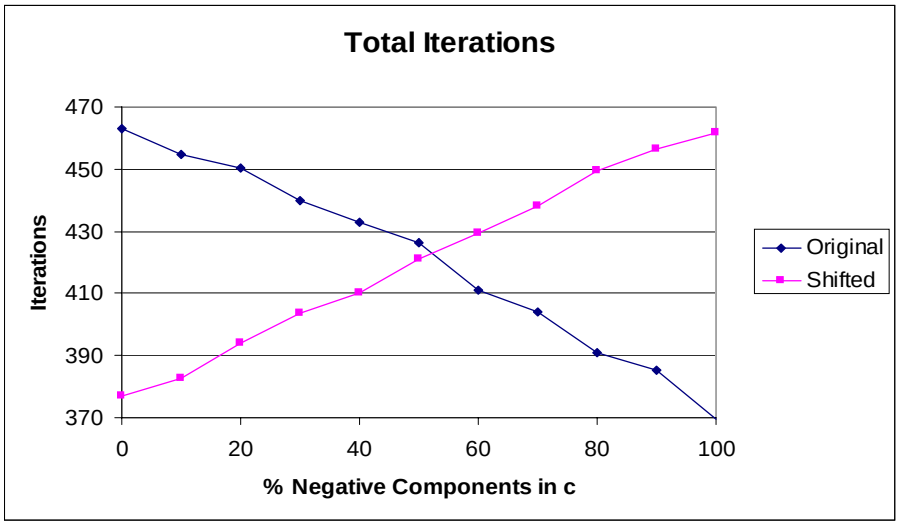

Figure 16: Effect of Percent of Negative Components of **c** on Total Iterations

more the number of negative components of **c**, the more the average number of iterations needed in phase 2 (and hence in total) to solve the shifted problem. By shifting the origin of the original LP, CPLEX solves problems differently. This is because of the way CPLEX chooses an initial starting basis and initial nonbasic values.

Moreover, in phase 1, CPLEX does not use the standard approach of minimizing the sum of the artificial variables. Rather, CPLEX minimizes a piecewise linear function consisting of the sum of the infeasibilities associated with a chosen initial—but not feasible—basic solution. This piecewise linear function depends on the current basis and its associated infeasibilities. Different CPLEX parameter settings can result in a different sequence of bases during phase 1.

4 Results of Applying Coordinate Transformations to the Problems in Netlib

Having addressed some of the implementation issues, the coordinate transformation is now applied to the problems in the Netlib collection, excluding the infeasible problems and the kennington problems. There are 93 Netlib problems of which five could not be read by CPLEX after decompression, namely, blend, dfl001, forplan, gfrd-pnc, and sierra.

After decompression, each of the 88 Netlib problem is solved by the following steps:

1. Modify the data for comparability.
   In an attempt to have the original and transformed problems be comparable, the following steps are performed.

   (a) Let CPLEX apply its own preprocessing procedure to the data.
   An alternative approach is to apply the coordinate transformation before CPLEX is allowed to do its own preprocessing on the resulting LP. If this strategy is used, however, a new transformed problem with different size and structure, which is created after preprocessing, does not "optimize" the benefits of the transformation. Another approach is not to use preprocessing for both the original problem and the transformed problem. However, this strategy will not take advantage of preprocessing, which often reduces solution time considerably.

   (b) Tighten bounds on all variables as much as possible.
   As observed in Section 3.1, the farther the starting point from the feasible region, the more work is expected in phase 1. Therefore, a special preprocessing code was written to find tighter bounds on variables by using one of the preprocessing techniques [see Brearley, Mitra, and Williams (1975)] prior to applying the coordinate transformation. To illustrate the tightening of bounds, consider the following LP:

$$\begin{array}{lccccc} \min & & & \mathbf{cx} & & \\ \text{s.t.} & \mathbf{b_l} & \leq & A\mathbf{x} & \leq & \mathbf{b_u} \\ & \mathbf{l} & \leq & \mathbf{x} & \leq & \mathbf{u} \end{array}$$

   where each component of $\mathbf{l}$ or $\mathbf{b_l}$ can be $-\infty$ and each component of $\mathbf{u}$ or $\mathbf{b_u}$ can be ∞. The bounds on the variables can be used to determine the smallest and largest possible value of constraint i by the following formulas:

$$\begin{aligned} \underline{b_i} &= \sum_{j \in P_i} A_{ij} l_j + \sum_{j \in N_i} A_{ij} u_j \\ \overline{b_i} &= \sum_{j \in P_i} A_{ij} u_j + \sum_{j \in N_i} A_{ij} l_j \end{aligned}$$

   where $P_i = \{j : A_{ij} > 0\}$ and $N_i = \{j : A_{ij} < 0\}$. Therefore, for any $\mathbf{x} \in [\mathbf{l}, \mathbf{u}]$, each constraint i is bounded by $\underline{b_i}$ and $\overline{b_i}$. If $b_{u_i} < \underline{b_i}$ or $b_{l_i} > \overline{b_i}$, then the problem is infeasible. If the problem cannot be detected as infeasible, then $\underline{b_i}$ or $\overline{b_i}$ can be used to tighten the bounds on the variables, as follows. For each constraint $i = 1, \ldots, m$,

   - For all $k \in P_i$, set $l_k = \max\left\{\frac{b_{l_i} - \overline{b_i}}{A_{ik}} + u_k, l_k\right\}$ and $u_k = \min\left\{\frac{b_{u_i} - \underline{b_i}}{A_{ik}} + l_k, u_k\right\}$.
   - For all $k \in N_i$, set $l_k = \max\left\{\frac{b_{u_i} - \underline{b_i}}{A_{ik}} + u_k, l_k\right\}$ and $u_k = \min\left\{\frac{b_{l_i} - \overline{b_i}}{A_{ik}} + l_k, u_k\right\}$.

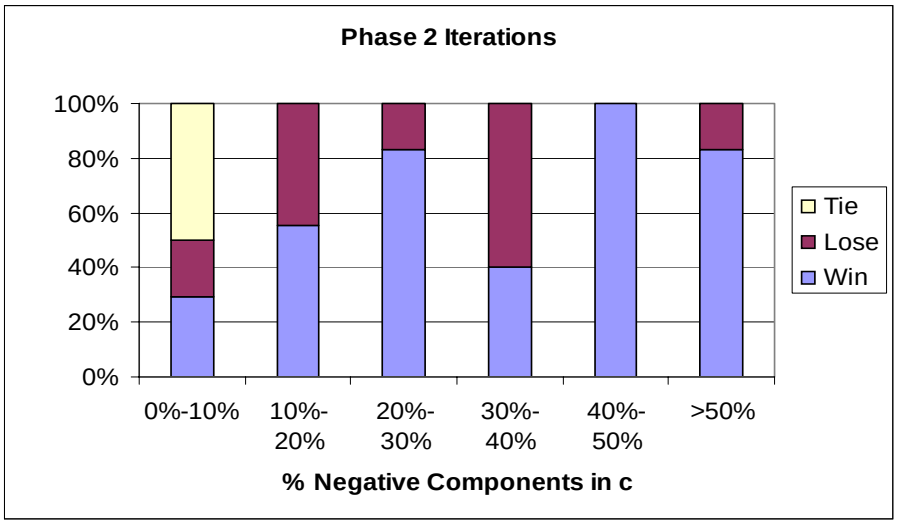

Figure 17: Effect of % of Negative Components of **c** on the % of Wins/Losses/Ties in Phase 2

Note that the foregoing formulas need only be applied when all values involved are finite. Repeating these computations numerous times may achieve tighter bounds. Therefore, this procedure is repeated ten times in all runs of Netlib problems.

(c) Shift all variables to have lower bounds of 0.
Due to the CPLEX issues described in Section 3.2, all variables are shifted so that their lower bounds are 0.

2. Solve the modified LP by setting all variables as nonbasic at their lower bounds of 0.
In order for CPLEX to start at the origin, all variables must be set as nonbasic at lower bounds of 0. If a variable does not have a finite lower bound but has a finite upper bound, this variable is set as nonbasic at upper bound. If a variable has neither a finite lower nor upper bound (a "free" variable), this variable is set as "super basic." By setting all variables as nonbasic, CPLEX is forced to use the slack or artificial variables in the starting basis.

3. Apply the coordinate transformation as described in Section 2.1 and solve the transformed LP by setting all variables as nonbasic at their lower bounds of 0.
Again, in order to start at the origin, all variables must be set as described above.

All runs were made in batch mode and default settings were used throughout in CPLEX.

The x-axis in Figure 17 represents the percent of negative components in the vector **c** after preprocessing and the y-axis represents the percent of wins, losses and ties in phase 2 to obtain the optimal solution for the transformed problems. Similar information pertaining to phase 1 and total number of iterations is shown in Figure 18 and Figure 19. As can be seen, the more the number of negative components of **c**, the higher the percent of winnings in phase 2 for the transformed problems. This is consistent with the simulation results in Section 2 and again justifies Conjecture 3 that the origin of the new coordinate system is close to an optimal solution when compared to other feasible points.

However, looking at Figure 18, the transformed problems are not performing as well as the

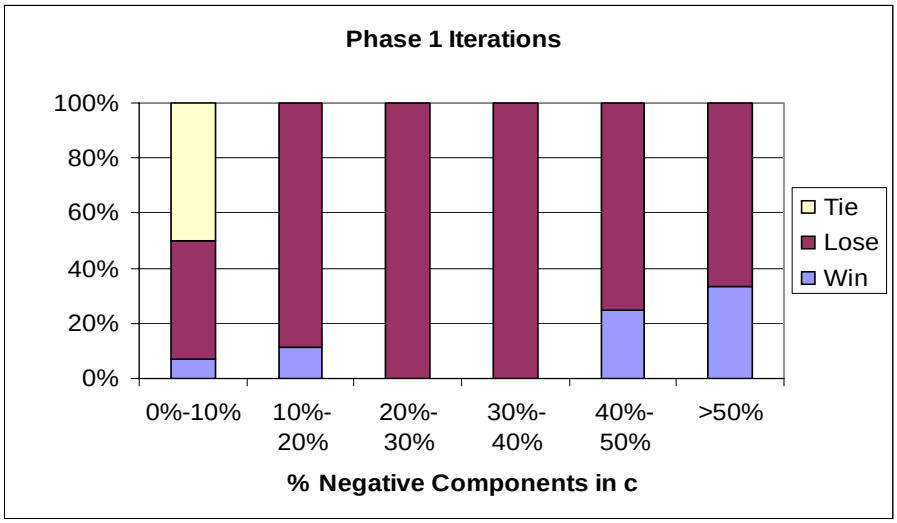

Figure 18: Effect of % of Negative Components of **c** on the % of Wins/Losses/Ties in Phase 1

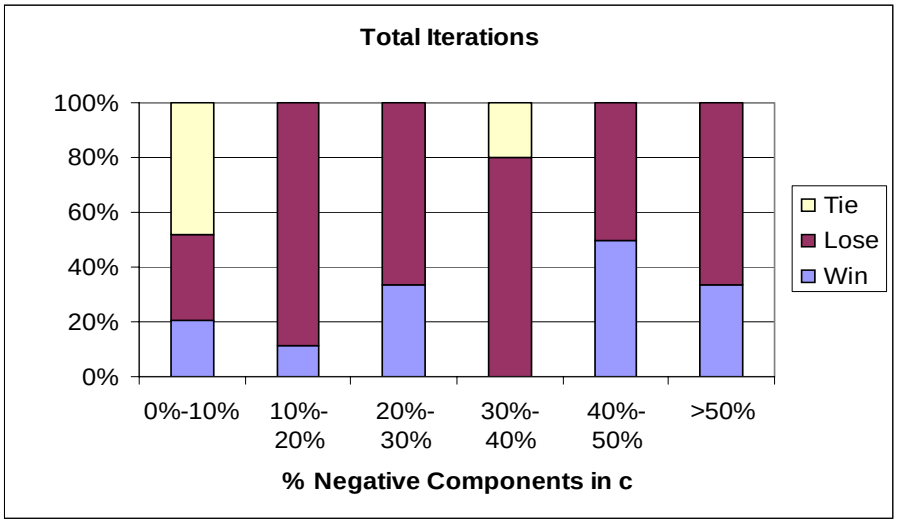

Figure 19: Effect of % of Negative Components of **c** on the % of Wins/Losses/Ties in Total Iterations

	Violated Constraints				
Phase 1		Fewer	Equal	More	Total
	Wins	0	2	6	8
	Ties	0	24	1	25
	Losses	1	2	52	55
	Total	1	28	59	88

Table 1: A Correlation of Violated Constraints and Phase 1 Iterations

original problems in phase 1 and hence in total iterations also. There are at least two possible reasons that create this anomaly with Netlib problems:

1. The feasible region is not be centered within the tightened lower and upper bounds.
   There are nine problems—namely, afiro, fit1d, fit2d, fit2p, kb2, recipe, sc105, sc205, and sc50a—that take no iterations in phase one to solve the original problems but take numerous iterations in phase 1 to solve the transformed problems. The problem israel takes one iteration to solve the original problem. The original problems sc50b and seba take five iterations and four iterations to solve, respectively. Therefore, some problems in the Netlib collection are not centered within the tightened lower and upper bounds.

2. There are more violated constraints at the starting point of the transformed LP than at the starting point of the original LP.
   As the number of violated constraints increases at the starting point of an LP, the amount of work in phase 1 is expected to increase. This is supported by the values in Table 1. Specifically, the columns in Table 1 indicate the number of problems in which there are fewer, equal, or more violated constraints at the starting point of the transformed LP than at the starting point of the original LP. Thus, the last row of Table 1 indicates that, of the 88 problems, only one had fewer violated constraints at the starting point of the transformed problem, 28 were tied, and 59 had more violated constraints. In contrast, the rows in Table 1 contain the number of wins, losses and ties in terms of iterations in phase 1 for the transformed LPs. Thus, the last column of Table 1 indicates that, of the 88 problems, the transformed problem won in phase 1 in only 8 problems, tied in 25, and lost in 55. As can be seen, the number of violated constraints at the starting point of the transformed LP is more than the number of violated constraints at the starting point of the original LP on $59/88 = 67\%$ of the Netlib problems. Correspondingly, the number of iterations in phase 1 for the transformed LP is more than the number of iterations in phase 1 for the original LP on $55/88 = 63\%$ of the Netlib problems. This confirms that if the number of violated constraints at the starting point of the transformed LP is more than the number of violated constraints at the starting point of the original LP, then finding an initial basic feasible solution for the transformed LP is expected to take more work than for the original LP.

Turning to the amount of time needed to solve the Netlib problems—as returned by the "times" C-library function—the results are quite different from those pertaining to the number of iterations. For example, the x-axis in Figure 20 represents the percent of negative components in the vector $\mathbf{c}$ after preprocessing and the y-axis represents the percent of wins, losses and (approximate) ties

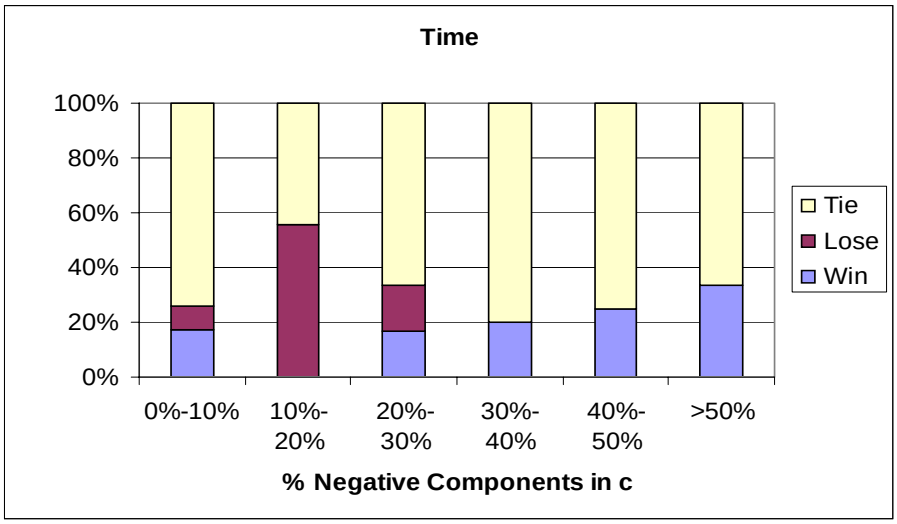

Figure 20: Effect of % of Negative Components of **c** on the % of Wins/Losses/Ties in Total Time

		Iterations			Time
Percent	Problems	Total	Phase 1	Phase 2	Total
$0\% - 10\%$	58	1.016	0.853	1.149	1.311
$10\% - 20\%$	9	0.924	0.653	1.204	0.922
$20\% - 30\%$	5	0.901	0.321	1.708	1.006
$30\% - 40\%$	5	0.871	0.542	1.052	1.188
$40\% - 50\%$	4	1.056	0.787	2.118	1.103
$> 50\%$	6	0.920	0.777	1.188	1.084
$0\% - 100\%$	87	0.987	0.775	1.228	1.221

Table 2: Average Ratios for Netlib Problems

in total time to obtain the optimal solution for the transformed problems compared to the original problems. As can be seen, the more the number of negative components of **c**, the higher the percent of winnings in total time for the transformed problems, and the transformed problems never lose in total time when **c** has at least 30% negative components. This would tend to indicate that, for the Netlib problems, phase 2 iterations require more time than phase 1 iterations. Thus, the time savings achieved by the fewer number of phase 2 iterations on the tranformed problems on average more than compensates for the increased number of phase 1 iterations.

It is important to note that in the simulations with the randomly generated problems presented in Section 2, both the number of problems and their sizes in each range of percent of negative components of **c** are controlled. The same is not true for the Netlib problems. In order to equalize the effect of the different numbers and sizes of the Netlib problems within each range of percent of negative components of **c**, four average ratios are presented in Table 2. For each problem in a given range, the ratio of the total iterations required to solve the original problem is divided by

		Iterations			Time
Percent	Problems	Total	Phase 1	Phase 2	Total
$0\% - 10\%$	10	1.192	0.666	1.798	2.878
$10\% - 20\%$	0	$--$	$--$	$--$	$--$
$20\% - 30\%$	1	1.191	0.000	1.378	1.300
$30\% - 40\%$	1	0.921	0.427	1.122	1.938
$40\% - 50\%$	1	1.097	0.879	1.457	1.410
$> 50\%$	2	1.270	1.308	1.108	1.252
$0\% - 100\%$	15	1.162	0.688	1.579	2.395

Table 3: Average Iteration Ratios for Winning Problems

the total iterations required to solve the transformed problems. The average of these ratios over all problems in the given range is computed and reported in Table 2. Similar average ratios for phase 1 iterations, phase 2 iterations, and total time are also given in that table. (Note that for the range 20%-30%, the problem afiro is omitted because this problem requires no iterations in phase 2 of the transformed problem and so that phase 2 ratio cannot be computed.) In summary, Table 2 contains, for each range of percent negative components in the objective function, the number of problems, the average ratios of total iterations, of phase 1 iterations, of phase 2 iterations, and of total time. The final row contains the corresponding average ratios over all 87 problems in Netlib.

If an average ratio in Table 2 is greater than 1, then it is more efficient to solve the transformed problems than the original problems, on average. Thus, for example, the phase 1 ratios being less than 1 indicate that solving the original problems on average requires fewer iterations in phase 1 than solving the transformed problems. Likewise, the phase 2 ratios being greater than 1 indicate that solving the transformed problems on average requires fewer iterations in phase 2 than solving the original problems.

Of particular interest is the fact that the time ratios in Table 2 are greater than 1 (except in the range 10%-20%), meaning that on average, solving the transformed problem is more efficient than solving the original problem. To gain further insights into this, Table 3 shows the average ratios over all problems where the transformed problems require less time to solve. Likewise, Table 4 shows the average ratios over all problems where the transformed problems require more time.

As can be seen from the phase 2 ratios in Table 3, the transformed problems on average require fewer iterations in phase 2, resulting in an overall savings in total iterations and total time. Of particular interest is the one problem in the $20\% - 30\%$ range. This problem requires no iterations in phase 1 for the original LP. However, the transformed problems take fewer iterations in phase 2 than the original problem and the resulting savings overcomes the losses in phase 1 iterations. Hence, the transformed problem is more efficient in both total iterations and total time than the original problem.

In contrast, the iteration ratios in Table 4 indicate that the savings in phase 2 iterations does not result in an overall savings in total iterations nor in total time. Tables 2-4 lead one to believe that, in general, when the transformed problem loses, it requires only slightly more time to solve than the original problem but when the transformed problem wins, it wins by a significant amount of time.

		Iterations			Time
Percent	Problems	Total	Phase 1	Phase 2	Total
$0\% - 10\%$	5	0.935	0.928	1.003	0.854
$10\% - 20\%$	5	0.850	0.643	1.117	0.859
$20\% - 30\%$	1	0.660	0.146	1.730	0.730
$30\% - 40\%$	0	$--$	$--$	$--$	$--$
$40\% - 50\%$	0	$--$	$--$	$--$	$--$
$> 50\%$	0	$--$	$--$	$--$	$--$
$0\% - 100\%$	11	0.871	0.727	1.121	0.845

Table 4: Average Iteration Ratios for Losing Problems

				Original				Transformed			
Problem	Cons.	Vars.	% N.C.	Total	Phase1	Phase2	Time	Total	Phase1	Phase2	Time
pilot	1203	3065	1.70%	7703	4711	2992	8.80	7947	4946	3001	9.13
greenbeb	1015	3039	5.56%	5154	2287	2867	0.75	5183	2659	2524	0.72
greenbea	1015	3048	5.71%	6056	2044	4012	0.72	6609	2228	4381	1.13
d2q06c	1855	4561	7.34%	10549	2698	7851	6.92	11927	2977	8950	7.47
pilot87	1809	4414	11.01%	9561	4102	5459	22.39	13330	9264	4066	26.88
80bau3b	1789	8510	12.64%	8856	1638	7218	0.44	10487	2045	8442	0.56
stocfor2	1070	1088	48.71%	1573	786	787	0.11	1434	894	540	0.08
degen3	1406	1721	64.03%	15333	13343	1990	4.42	11595	9622	1973	3.39

Table 5: Big Netlib Problems

For "large" Netlib problems that have more than 1000 constraints, Table 5 shows the total number of constraints and variables, the percent of negative components of the vector $\mathbf{c}$ in the original problems (after preprocessing), total iterations, phase 1 iterations, phase 2 iterations, and total time for both original and transform problems. As can be seen, the transformed problems require less time than the original problems when there are a sufficient number of negative components in the $\mathbf{c}$ vector.

In summary, the foregoing results indicate that although the transformed Netlib problems generally require more phase 1 and total iterations than the original problem, solving the transformed problem is more efficient, regardless the size, on average with regard to total time due to the savings of iterations in phase 2. Also, when the number of negative components in the minimization objective function of the original problem exceeds about 20%, solving the transformed problem wins with regard to total time in a significant fraction of the Netlib problems.

Conclusion and Extensions

This paper addresses the open question of where phase 1 of the simplex algorithm terminates on the feasible region of an LP. Computational results on randomly-generated problems provide convincing evidence that phase 1 terminates at a feasible point that is geometrically close to the

initial values of the variables when compared to other feasible points and also that such an initial feasible point reduces the number of iterations required in phase 2. This observation is then used to apply a coordinate transformation to an original bounded LP so that the origin of the resulting transformed problem, which is assumed to be the starting point for phase 1, is geometrically close to the optimal solution when compared to other feasible points. Computational results on randomly-generated problems indicate that the benefits of solving the transformed problem—in terms of the reduced number of iterations needed to solve the problem—increase as the number of negative components in the minimization objective function increases. Similar benefits in phase 2 and total time are realized when the coordinate transformation is applied to solve the problems in Netlib. On average, solving the transformed Netlib problems is more efficient than solving the original problem with regard to total time due to the savings of iterations in phase 2.

Based on the results presented here, the closer the starting point is to the optimal solution, the more efficiently the simplex algorithm is likely to be in obtaining the optimal solution. This suggests that it may be worthwhile trying to find better starting points for the simplex algorithm. One possibility is to find tighter lower and upper bounds. Another possiblity is to consider variables whose objective function coefficients are 0. According to the formula in (2), each such variable could be set equally well to its upper or lower bound. In fact, it may be better to set such a variable to an initial value that is half way between its lower and upper bound. To do so would require a phase 1 procedure that allows each nonbasic variable to start at a value that is strictly between its upper and lower bound. Such a phase 1 procedure is easy to develop.

References

R. E. Bixby, "Implementing the Simplex Method: The Initial Basis", *ORSA Journal on Computing*, **4**:267-284, 1992.

A. L. Brearley, G. Mitra, and H.P. Williams, "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Method", *Mathematical Programming*, **8**:54-83, 1975.

D. M. Carstens, "Crashing Techniques", in: W. Orchard-Hays, *Advanced Linear-Programming Computing Techniques*, McGraw-Hill, New York, 131-139, 1968.

N. I. M. Gould and J. K. Reid, "New Crash Procedures for Large System of Linear Constraints", *Mathematical Programming*, **45**:475-501, 1989.

N. Karmarkar, "A New Polynomial Time Algorithm for Linear Programming", *Combinatorica*, **4**:373-395, 1984.

I. Maros, "A General Phase-I Method in Linear Programming", *European Journal of Operation Research*, **23**:64-77, 1986.