Comput Optim Appl (2012) 52:345–371
DOI 10.1007/s10589-011-9420-4

Scaling linear optimization problems prior to application of the simplex method

Joseph M. Elble · Nikolaos V. Sahinidis

Received: 24 April 2010 / Published online: 13 July 2011

Abstract The scaling of linear optimization problems, while poorly understood, is definitely not devoid of techniques. Scaling is the most common preconditioning technique utilized in linear optimization solvers, and is designed to improve the conditioning of the constraint matrix and decrease the computational effort for solution. Most importantly, scaling provides a relative point of reference for absolute tolerances. For instance, absolute tolerances are used in the simplex algorithm to determine when a reduced cost is considered to be nonnegative. Existing techniques for obtaining scaling factors for linear systems are investigated herein. With a focus on the impact of these techniques on the performance of the simplex method, we analyze the results obtained from over half a billion simplex computations with CPLEX, MINOS and GLPK, including the computation of the condition number at every iteration. Some of the scaling techniques studied are computationally more expensive than others. For the Netlib and Kennington problems considered herein, it is found that on average no scaling technique outperforms the simplest technique (equilibration) despite the added complexity and computational cost.

Keywords Scaling · Linear optimization · Condition number · Simplex method

Electronic supplementary material The online version of this article (doi:10.1007/s10589-011-9420-4) contains supplementary material, which is available to authorized users.

J.M. Elble
Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: elble@uiuc.edu

N.V. Sahinidis (✉)
Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: sahinidis@cmu.edu

1 Introduction

The main objective of scaling a linear optimization problem is to improve the conditioning of the constraint matrix and ultimately the accuracy of the solution. However, there are many potential benefits of scaling a linear optimization problem. In particular, scaling may decrease the number of iterations required to solve the problem. In addition, scaling provides a reference point for the magnitude of static tolerances that are an integral part of practical linear optimization solvers.

The means by which scaling is generally believed to improve numerical behavior is by creating row and column scaling factors whose application to the constraint matrix leads to nonzero numerical values of similar magnitude. Although matrices with nonzero numerical values of similar magnitude are generally thought to define good scaling [14, 17, 21, 22, 24], it remains unclear just how well scaled such a matrix is.

Tomlin [24] presented a thesis on the rationale behind scaling linear optimization problems and some computational results for arithmetic mean, geometric mean, equilibration, Curtis and Reid [6], Fulkerson and Wolfe [12] scaling techniques, and various combinations. Larsson [17] expanded on Tomlin's study by including entropy (attributed to Dantzig and Erlander in [17]), L_p–norm [14], and de Buchet [8] scaling models. Larsson's results on randomly generated linear optimization problems suggest that it may be possible to use scaling to reduce the number of simplex iterations. In addition, Larsson performed a small study on the condition number of the constraint matrix before and after scaling, and suggested that the entropy model is often able to improve the conditioning of the randomly generated linear optimization problems. The results of Larsson's paper were so promising that he wrote:

> We have experimentally shown that it may be possible to significantly decrease the number of iterations in the simplex method by prescaling. The results are so promising that it would be of interest to perform a computational study with large-scale real-world linear programs from different applications.

Tomlin's study [24] was limited by the quantity and diversity of real-world test problems available to researchers in the mid 1970s. The study concentrated on six test problems of varying sizes. The scaling paper written by Larsson [17] used more test problems but these were generated with Gaussian random matrices of varying sparsity. It is now well understood that Gaussian random constraint matrices are unlikely to have any ill-conditioned bases (cf. [16]) and may, therefore, not be representative of real problems. To date, there has been no published systematic study of scaling algorithms on a large collection of test problems. One of the major goals of this paper is to fill this void in the linear optimization literature.

Both Tomlin [24] and Larsson [17] discussed the objective of using scaling to reduce the number of simplex iterations required to solve a linear optimization problem. Using sparse linear optimization problems from the Netlib and Kennington libraries, in Sect. 6, we find through extensive computational experimentation that, on average, none of the scaling algorithms proposed to date is able to reduce the number of required simplex iterations.

In addition to the goal of reducing the number of iterations, scaling is thought to improve the "numerical behavior" of the simplex algorithm. While scaling is certainly able to provide a reference point for the magnitude of absolute tolerances, it is unclear whether it improves the "numerical behavior" of the algorithm. In the face of round-off error, a properly designed simplex-based code will contain a dozen or more absolute tolerances [24]. These tolerances, for instance, are responsible for determining when a linear optimization problem is infeasible (taking into account possible rounding error). A tolerance is also used to determine eligible pivot elements, i.e., given round-off error, the tolerance is used to determine if an element is sufficiently different from zero. Reduced costs are also subject to absolute tolerances. A reduced cost is said to be negative, if and only if it is less than $-\varepsilon$, where ε is the appropriate tolerance.

Again, based on the results presented in Sect. 6 and an example presented in Sect. 5, it is questionable whether scaling improves the "numerical behavior" of the simplex algorithm. The computational results suggest that scaling can cause an increase in the average condition number of the square basis matrices B in the simplex algorithm, where the condition number is defined as $\kappa(B) = \|B\|\|B^{-1}\|$. Such an increase will necessarily affect the accuracy of solving linear systems involving B and B^T, thereby degrading the accuracy of the potential pivot elements and the reduced costs. In addition, the increased condition number of a given basis (or series of bases) may necessitate a more frequent refactorization of the basis. The example offered in Sect. 5 illustrates the potential increase in the average condition number of the square basis matrices B due to scaling.

After mathematical preliminaries are provided in Sect. 2, the remainder of the paper begins with a review of scaling techniques in Sects. 3 and 4. We classify and interpret existing techniques based on whether or not they represent approximate solutions to an optimization model derived from the data of the original problem. Basic scaling techniques are presented in Sect. 3, followed by model-based scaling techniques in Sect. 4. Every scaling technique presented herein is described using a Gauss-Seidel iterative scheme, except for the composite-Jacobi binormalization algorithm presented in Sect. 4.4. The choice of these iterative schemes is guided by historical considerations in the former cases and by efficiency concerns in the case of binormalization. A small problem is provided in Sect. 5 that illustrates the complexity involved in properly scaling a linear optimization problem. This example is followed by the computational results compiled from the over half a billion simplex computations with CPLEX, MINOS and GLPK, in Sect. 6. Given the large volume of data, many of our results are presented in the online resource. Concluding remarks are offered in Sect. 7.

2 Mathematical and notational preliminaries

Before the scaling techniques and models are described, some necessary notation is introduced. Let A be an $m \times n$ real matrix. It is assumed that a suitable presolve routine has removed all empty rows and columns from A. When results are discussed in Sect. 6, "with presolve" refers to the use of CPLEX's Presolve routine and "without

presolve" refers to the removal of empty rows and columns only. That is, an experiment "without presolve" involves setting the problem up for scaling and does not involve more advanced presolve techniques.

Let $N_i = \{j \mid a_{ij} \neq 0\}$ for $i = 1, \ldots, m$, $M_j = \{i \mid a_{ij} \neq 0\}$ for $j = 1, \ldots, n$, and $\overline{Z} = \{(i, j) \mid a_{ij} \neq 0\}$. Let n_i and m_j be the cardinality of the sets N_i and M_j, respectively. Let r be an m-vector and s be an n-vector, where r_i is the scale for row i and s_j is the scale for column j. The scaled matrix is expressed as $X = RAS$, where $R = \text{diag}(r_1 \ldots r_m)$ and $S = \text{diag}(s_1 \ldots s_n)$. The scaling methods presented in this paper are all of the same type, in that they all involve a scaling of the rows and then the columns. The process by which these row and column scales are applied is iterative. Let $X^{(0)} = A$. Then, each iteration is given by

$$X^{(k+1/2)} = R^{(k+1)} X^{(k)},$$
$$X^{(k+1)} = X^{(k+1/2)} S^{(k+1)},$$

where

$$R = \prod_{k=1}^{t} R^{(k)},$$
$$S = \prod_{k=1}^{t} S^{(k)},$$

and t is the number of iterations needed to obtain an optimal scaling. The term *optimal scaling* refers to the scaled matrix to which a specific scaling model converges. This is not to be confused with *the problem of optimal scaling* [1], which refers to finding R and S such that $\kappa(RAS)$, the condition number of RAS, is minimized. The formulation of this optimization problem is

$$\begin{array}{lll} \text{(S)} & \min \kappa(RAS) & \\ & \text{s.t.} \quad r_i > 0 & \forall i \in [1, m] \\ & \qquad s_i > 0 & \forall i \in [1, n], \end{array}$$

with a suitable definition of $\kappa(RAS)$ for matrices that are not necessarily square or of full rank. The condition number of RAS could be computed from its singular value decomposition [13], but this is expensive for sparse matrices. No scaling algorithm solves this idealistic goal.

In practice, once the scaling matrices R and S have been obtained by a scaling technique, it is sometimes beneficial to use the nearest power of two rather than the scaling factors obtained by the algorithm. This is referred to as *power-of-two scaling* and affords exact machine representation of the resultant matrix elements, thus avoiding round-off error. For instance, a constraint matrix with a large number of $+1$ and -1 coefficients may benefit from power-of-two scaling. Scaling two rows by very slightly different scaling factors will cause these coefficients to slightly differ from one another. In factorization algorithms, these coefficients (left unscaled) might result in cancellation. However, if scaling factors were applied that cause these numbers to differ in magnitude by a very small amount, this cancellation will not occur.

Indeed, Tomlin [24] found that scaling might also bias the pivot choice in such a way as to reduce cancellation. GLPK [19] recently afforded users access to power-of-two scaling in Library Package 4.31. In addition, dividing by a power of two amounts to a mere shift operation (an increase/decrease in the exponent of a floating-point number), thus improving the computational efficiency and identification of termination. Whether or not power-of-two scaling is used should depend on the characteristics of the nonzero elements in the problem.

We assume, without loss of generality, that the elements of A are nonnegative. This assumption is made merely to simplify the presentation of the subsequent scaling techniques. That is, the nonnegativity assumption permits the presentation of formulae without the otherwise obligatory absolute values.

3 Basic scaling techniques

3.1 Equilibration

Equilibration scaling and the effect of equilibration on the condition number of square matrices was studied by van der Sluis [25, 26]. In equilibration scaling, each row is scaled to make its largest nonzero entry of magnitude one. This is followed by a similar column scaling. It is well known that equilibration can generate small round-off errors. In other scaling procedures, it is common for solvers to offer a power-of-two scaling option to avoid round-off error. While it is possible to use power-of-two scaling in conjunction with equilibration, such a scaling would not by definition lead to an equilibration scaling. Using power-of-two scaling, there is no way to ensure that the largest nonzero entry is of magnitude one.

3.2 Geometric mean

Geometric mean scaling is a technique designed to decrease the variance between the nonzeros in the matrix. In geometric mean scaling, the row scaling factor r_i is calculated as follows:

$$r_i^{(k+1)} = \left(\max_{j \in N_i} x_{ij}^{(k)} \min_{j \in N_i} x_{ij}^{(k)} \right)^{-1/2}.$$

This row scaling procedure is followed by a similar column scaling. The column scaling factor s_j is

$$s_j^{(k+1)} = \left(\max_{i \in M_j} x_{ij}^{(k+1/2)} \min_{i \in M_j} x_{ij}^{(k+1/2)} \right)^{-1/2}.$$

These scaling factors are not the true geometric mean, rather an approximation suggested by Tomlin [24].

3.3 Arithmetic mean

Like geometric mean, arithmetic mean also aims to decrease the variance between the nonzeros in the matrix. The row scaling factor in arithmetic mean scaling is the inverse of the mean of all nonzero elements for each row, or the nearest power of two. Like the other scaling techniques presented thus far, a similar column scaling follows the row scaling. The row and column scales according to this method are

$$r_i^{(k+1)} = \left(n_i / \sum_{j \in N_i} x_{ij}^{(k)} \right), \quad \forall i, \quad \text{and}$$

$$s_j^{(k+1)} = \left(m_j / \sum_{i \in M_j} x_{ij}^{(k+1/2)} \right), \quad \forall j,$$

respectively.

3.4 Combination

It is quite common to see either geometric and arithmetic mean scaling followed by equilibration.

3.5 Dynamic scaling: IBM's MPSX

Originally proposed by Benichou et al. [2], the following scaling technique was later adopted by IBM for use in IBM's MPSX (Mathematical Programming System Extended), which was developed to solve linear optimization problems. This dynamic scaling algorithm utilized a combination of geometric mean and equilibration scaling. Geometric mean scaling is performed four times or until

$$\frac{1}{|\overline{Z}|} \left[\sum_{(i,j) \in \overline{Z}} a_{ij}^2 - \left(\sum_{(i,j) \in \overline{Z}} |a_{ij}| \right)^2 \Big/ |\overline{Z}| \right] < \varepsilon,$$

whichever is fewer, where $|\overline{Z}|$ is the number of nonzero entries in A and ε is a tolerance, typically set below ten, often at four. After the geometric scaling phase is complete and the variance between the nonzeros in the matrix has been decreased sufficiently, equilibration scaling is performed.

4 Model-based scaling techniques

4.1 L_p-norm scaling model

The L_p-norm scaling model is formulated as

$$\min_{r,s>0} \left\{ \sum_{(i,j) \in \overline{Z}} \left| \log \left(a_{ij} r_i s_j \right) \right|^p \right\}^{1/p}.$$

The classical version of this model was introduced by Hamming [14] and has $p = 1$.

By minimizing the sum of the absolute value of the logarithm of each scaled nonzero, this scaling model seeks to minimize the relative divergence of the nonzero elements of the problem from one. The sensitivity to the largest and smallest absolute nonzero matrix entries depends on p, where p is a positive integer. An equivalent unconstrained optimization problem is

$$\min_{\rho,\sigma>0} \left\{ \sum_{(i,j)\in\overline{Z}} |\alpha_{ij} + \rho_i + \sigma_j|^p \right\}^{1/p},$$

where $\alpha_{ij} = \log(a_{ij})$, $\rho_i = \log(r_i)$ and $\sigma_j = \log(s_j)$. If ρ_i and σ_j are rounded to the nearest integer and the log base two is used to calculate α_{ij}, ρ_i, and σ_j, then power-of-two multipliers are obtained. The optimization problem above is equivalent to the problem of finding a best L_p-approximate solution to the over-determined system of linear equations $\rho_i + \sigma_j = -\alpha_{ij}$. This problem was studied in [5] for the cases $p = 1$ and $p = \infty$, and in [7] for the case $p = 2$. The focus of the remainder of this section will be on these three cases.

4.1.1 L_1-norm scaling model

In [17], Larsson describes the optimal scaling for the case $p = 1$. The necessary and sufficient conditions for optimality of X are given below for the L_1-norm scaling model.

- Necessary Condition (L_1-norm): Let $k_i^+ = |\{j \mid x_{ij} > 1\}|$ for $i = 1, \ldots, m$, and $k_i^- = |\{j \mid 0 < x_{ij} < 1\}|$ for $i = 1, \ldots, m$; and let l_j^+ and l_j^- be similarly defined for the columns. The matrix X is an optimal solution if $k_i^+ = k_i^-$, $\forall i$ and $l_j^+ = l_j^-$, $\forall j$.
- Sufficient Condition (L_1-norm): Let X be an optimal solution to the L_1-norm scaling model. Let $k_i^0 = |\{j \mid x_{ij} = 1\}|$, $\forall i$ and $l_j^0 = |\{i \mid x_{ij} = 1\}|$, $\forall j$. Then $k_i^0 \geq |k_i^+ - k_i^-|$ and $l_j^0 \geq |l_j^+ - l_j^-|$ hold for all i and j.

Prior to the introduction of these optimality criteria in [17], the literature on scaling had not yet addressed such criteria for the L_1-norm model. For further insight into their origin, the reader is referred to [17].

The L_1-norm scaling model's row and column scaling factors at each iteration are chosen based on the necessary and sufficient conditions provided. The optimality conditions are satisfied by dividing each row and column by the median of the absolute value of the nonzero entries:

$$r_i^{k+1} = 1/\text{median}\left\{x_{ij}^{(k)} | j \in N_i\right\}, \quad \forall i, \quad \text{and}$$

$$s_j^{k+1} = 1/\text{median}\left\{x_{ij}^{(k+1/2)} | i \in M_j\right\}, \quad \forall j.$$

4.1.2 L_2-norm scaling model

A model similar to the L_2-norm scaling model was originally proposed by Hamming [14] and later enhanced by Curtis and Reid [6], who suggested using a specialized conjugate-gradient method to solve for the optimal scaling factors. Curtis and Reid's approach allowed for one or more matrix entries to be zero, while Hamming's closed form solution for the scaling factors required that all matrix entries be nonzero. This model gained further popularity after the experiments in [24] suggested that the Curtis-Reid scaling method was generally superior to the earlier Fulkerson and Wolfe's scaling method [12], which is equivalent to the L_∞-norm model discussed below. The L_2-norm model is stated as

$$\min_{r,s>0} \left\{ \sum_{(i,j)\in\overline{Z}} \left\{ \log\left(a_{ij} r_i s_j\right) \right\}^2 \right\}^{1/2},$$

which is equivalent to

$$\min_{r,s>0} \sum_{(i,j)\in\overline{Z}} \left\{ \log\left(a_{ij} r_i s_j\right) \right\}^2.$$

The solution to this model is optimal if and only if the product of the nonzero matrix entries in each row and column of $X = RAS$ equals one.

The row and column scaling factors for the L_2-norm model are realized by taking the reciprocal of the geometric mean of the nonzero elements:

$$r_i^{(k+1)} = 1 \Bigg/ \left(\prod_{j\in N_i} x_{ij}^{(k)} \right)^{1/n_i}, \quad \forall i, \quad \text{and}$$

$$s_j^{(k+1)} = 1 \Bigg/ \left(\prod_{i\in M_j} x_{ij}^{(k+1/2)} \right)^{1/m_j}, \quad \forall j.$$

It is important to note that, while the scaling factors for the L_2-norm model are the inverses of the geometric means, the geometric mean scaling method presented in Sect. 3.2 does not have the same scaling factors, rather those suggested by Tomlin [24]. Next, we shall see that the L_∞-norm model corresponds to the geometric mean method, with the exception that L_∞-norm possesses a termination criterion based on the optimization model and its optimality conditions.

The scaling factors and optimality conditions are presented in this section, while detailed derivations are offered in [9].

4.1.3 L_∞-norm scaling model

The L_∞-norm model is formulated as

$$\min_{r,s>0} \left\{ \max_{(i,j)\in\overline{Z}} \left| \log(a_{ij} r_i s_j) \right| \right\},$$

and can be shown to be equivalent to

$$\min_{r,s>0} \frac{\max_{(i,j)\in\overline{Z}}(a_{ij}r_is_j)}{\min_{(i,j)\in\overline{Z}}(a_{ij}r_is_j)}.$$

This reformulation of the L_∞-norm model was first investigated in [12].

In the case of the L_∞-norm model, [17, 23, 27] described the following necessary and sufficient condition for optimality. Let $X = RAS$ be a solution to the L_∞-norm model and $w = \max_{(i,j)\in\overline{Z}} |\log x_{ij}|$. Then X is optimal if and only if there exists a cycle $\{(i_1, j_1), (i_1, j_2), (i_2, j_2), \ldots, (i_{t-1}, j_t), (i_t, j_t), (i_t, j_1), (i_1, j_1)\}$ of entries in X alternately taking the values $\exp(w)$ and $\exp(-w)$.

Since X is optimal for the L_∞-norm model if and only if it contains a cycle of alternating $\exp(w)$ and $\exp(-w)$, the row and column scaling factors should be based on the reciprocal of the product of the smallest and largest absolute elements in a given row or column. The row and column scales, respectively, are

$$r_i^{(k+1)} = 1 \Big/ \left\{ \left(\max_{j\in N_i} x_{ij}^{(k)} \right) \left(\min_{j\in N_i} x_{ij}^{(k)} \right) \right\}^{1/2}, \quad \forall i$$

$$s_j^{(k+1)} = 1 \Big/ \left\{ \left(\max_{i\in M_j} x_{ij}^{(k+1/2)} \right) \left(\min_{i\in M_j} x_{ij}^{(k+1/2)} \right) \right\}^{1/2}, \quad \forall j.$$

At this point, it is clear that one iteration of the L_∞-norm model corresponds to geometric mean scaling as described in Sect. 3.2.

4.2 The entropy scaling model

The first use of the entropy scaling model is attributed to Dantzig and Erlander by Larsson [17]. This technique seeks to identify a scaling X, with all $x_{ij} \neq 0$ of order unity, by solving the following model:

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in\overline{Z}} x_{ij} \left(\log \left(x_{ij}/a_{ij} \right) - 1 \right) \\
\text{s.t.} \quad & \sum_{j\in N_i} x_{ij} = n_i \quad i = 1, \ldots, m \\
& \sum_{i\in M_j} x_{ij} = m_j \quad j = 1, \ldots, n \\
& x_{ij} \geq 0 \quad \forall (i,j) \in \overline{Z}.
\end{aligned}$$

It follows from the model itself that the optimality conditions are the prescribed multiplicative structure $X = RAS$. This is a direct consequence of the chosen objective function and constraints. A proof of the optimality conditions for this model, as well as many other models presented herein, is contained in [9].

It is possible to use any algorithm that solves entropy problems in order to obtain the entropy scaling matrices. However, it is recommended, if for no purpose

other than uniformity, that one should apply iteratively the following row and column scales:

$$r_i^{(k+1)} = n_i / \sum_{j \in N_i} x_{ij}^{(k)}, \quad \forall i, \quad \text{and}$$

$$s_j^{(k+1)} = m_j / \sum_{i \in M_j} x_{ij}^{(k+1/2)}, \quad \forall j.$$

For more details, the reader is referred to [17].

4.3 The de Buchet scaling model

The de Buchet model, like the scaling models presented before it, is based on the relative divergence from one:

$$\min_{r,s>0} \left[\sum_{(i,j) \in \overline{Z}} \left\{ a_{ij} r_i s_j + 1 / \left(a_{ij} r_i s_j \right) \right\}^p \right]^{1/p}.$$

As in the L_p-norm model, a positive integer p can be varied to make the model more and less sensitive to the extreme values. This model was introduced in [8] for the case $p = 2$. In comparison to the L_p-norm models, the de Buchet scaling models are easier to implement because of the relative ease with which one can check the optimality conditions for the $p = 1$ and $p = 2$ cases.

While the $p = \infty$ case of L_p-norm and de Buchet scalings are identical, this is not true of the other two cases discussed. L_p-norm and de Buchet's objective functions are minimized when their arguments (the nonzero elements of constraint matrix) are one. Consider the case of $p = 1$. The objective function of L_1-norm scaling is the sum of the absolute value of the logarithm of the scaled nonzeros. The absolute value of the logarithm is minimized at one. Similarly, de Buchet's objective function at $p = 1$ is the sum of the scaled nonzeros and their inverses. The sum of a positive real number and its inverse is minimized at one. The variation in the objective function between L_p-norm and de Buchet models results in different optimality conditions and different iterates.

4.3.1 de Buchet $p = 1$

The optimality conditions for the case $p = 1$ were first given in [17] and state that R and S is an optimal scaling if and only if $X = RAS$ satisfies $\sum_{j \in N_i} x_{ij} = \sum_{j \in N_i} 1/x_{ij}, \forall i$ and $\sum_{i \in M_j} x_{ij} = \sum_{i \in M_j} 1/x_{ij}, \forall j$.

The iterative updates in de Buchet scaling models are suggested by their optimality conditions. These row and column scaling factors are given by

$$r_i^{(k+1)} = \left\{ \left(\sum_{j \in N_i} 1/x_{ij}^{(k)} \right) \Big/ \left(\sum_{j \in N_i} x_{ij}^{(k)} \right) \right\}^{1/2}, \quad \forall i$$

$$s_j^{(k+1)} = \left\{ \left(\sum_{i \in M_j} 1/x_{ij}^{(k+1/2)} \right) \Big/ \left(\sum_{i \in M_j} x_{ij}^{(k+1/2)} \right) \right\}^{1/2}, \quad \forall j.$$

For derivations of the optimality conditions, the reader is referred to [9].

4.3.2 de Buchet $p = 2$

The optimality conditions for the case $p = 2$ were first given in [8] and state that R and S is an optimal scaling if and only if $X = RAS$ satisfies $\sum_{j \in N_i} (x_{ij})^2 = \sum_{j \in N_i} 1/(x_{ij})^2, \forall i$ and $\sum_{i \in M_j} (x_{ij})^2 = \sum_{i \in M_j} 1/(x_{ij})^2, \forall j$.

Again, the optimality conditions suggest the following appropriate row and column scaling factors for the de Buchet $p = 2$ scaling model:

$$r_i^{(k+1)} = \left\{ \left(\sum_{j \in N_i} 1/\left(x_{ij}^{(k)}\right)^2 \right) \Big/ \left(\sum_{j \in N_i} \left(x_{ij}^{(k)}\right)^2 \right) \right\}^{1/4}, \quad \forall i, \quad \text{and}$$

$$s_j^{(k+1)} = \left\{ \left(\sum_{i \in M_j} 1/\left(x_{ij}^{(k+1/2)}\right)^2 \right) \Big/ \left(\sum_{i \in M_j} \left(x_{ij}^{(k+1/2)}\right)^2 \right) \right\}^{1/4}, \quad \forall j.$$

For derivations of the optimality conditions, the reader is referred to [9].

4.3.3 de Buchet $p = \infty$

The de Buchet model for the $p = \infty$ model is equivalent to the L_∞-norm model [17], and is formulated as

$$\min_{r,s>0} \left[\max_{(i,j) \in \overline{Z}} \left\{ a_{ij} r_i s_j + 1/a_{ij} r_i s_j \right\} \right].$$

Given this model's equivalence to the L_∞-norm model, it will not be discussed further.

4.4 Non-square binormalization (NBIN)

Binormalization algorithms seek to find R and S such that $\kappa(RAS)$, the condition number of RAS, is likely to be minimized, i.e., they seek an approximate solution of model S. The principal motivation behind binormalization algorithms is the Forsythe-Straus theorem [11], which implies that matrices satisfying a certain property and that have diagonal entries of order unity also have minimum condition number with respect to all their scaled counterparts. In order to obtain matrices of minimum condition number, the binormalization algorithm presented in [18] scales all the diagonal entries of AA^T to one. In general, the Non-square BINormalization (NBIN) algorithm has relatively fast convergence (usually converges in the first few iterations) and is effective at reducing the condition number of a rectangular matrix. However,

each iteration is time consuming compared to the scaling techniques presented in the previous sections. The NBIN algorithm presented in [18] is based on a composite-Jacobi relaxation. While slower than many of the scaling techniques presented herein, this scaling algorithm is made competitive for dense linear systems using a graphics processing unit in [10]. In [4], the author reviews binormalization algorithms and develops approximate binormalization algorithms that access a matrix only by matrix-vector products.

The remainder of this section is used to present an abbreviated derivation of the binormalization algorithm. The algorithm is initially formulated for a real symmetric $n \times n$ matrix A. For such a linear system, the binormalization algorithm seeks a scaled matrix $X = RAR$ that satisfies

$$\rho \equiv \sum_{j=1}^{n} X_{ij}^2 = \sum_{j=1}^{n} r_i^2 A_{ij}^2 r_j^2, \quad i = 1, \ldots, n, \tag{1}$$

where ρ is a constant in $\mathbb{R}^+$. Let $\beta_i = \sum_{\{j|r_j>0\}} A_{ij}^2 r_j^2$, $i = 1, \ldots, n$ and $B = \operatorname{diag}(\beta_1, \ldots, \beta_n)$. The solution r of the system

$$Br = be, \tag{2}$$

where $b > 0$ is arbitrary and $e = (1, \ldots, 1)^T$, is the solution to (1). Equivalently,

$$\left(I_n - \frac{1}{n} ee^T\right) Br = be - \frac{1}{n} ee^T be = 0. \tag{3}$$

If a positive solution exists to (3), then A is said to be scalable. Livne and Golub proposed using a Gauss-Seidel-Newton iterative scheme to solve (3).

In this paper, attention is necessarily focused on their generalization of the binormalization algorithm to a general $m \times n$ matrix A. The scaled matrix X has the form $X = RAS$, where $R = \operatorname{diag}(r_1, \ldots, r_m)$ and $S = \operatorname{diag}(s_1, \ldots, s_n)$. Let $B_{ij} = A_{ij}^2$, $\lambda_i = r_i^2$, $\mu_i = s_i^2$, $\beta_i = \sum_{j=1}^{n} B_{ij}\mu_j$ and $\gamma_j = \sum_{i=1}^{m} B_{ij}\lambda_i$. In addition, define $B(\mu) = \operatorname{diag}(\beta_1(\mu), \ldots, \beta_m(\mu))$ and $C(\lambda) = \operatorname{diag}(\gamma_1(\lambda), \ldots, \gamma_n(\lambda))$. The solution (r, s) to the asymmetric binormalization equation also solves the system

$$B(\mu)\lambda = be \quad \text{and} \quad C(\lambda)\mu = ce \tag{4}$$

for any $b, c > 0$. Notice that system (4) is analogous to system (2). Again, equivalently,

$$\left(I_m - \frac{1}{m} \widetilde{e}\,\widetilde{e}^T\right) B(\mu)\lambda = 0 \quad \text{and} \quad \left(I_n - \frac{1}{n} ee^T\right) C(\lambda)\mu = 0 \tag{5}$$

where $e = (1, \ldots, 1)^T$ is an n-vector and $\widetilde{e} = (1, \ldots, 1)^T$ is an m-vector. Notice again that system (5) is analogous to system (3). Livne and Golub propose a composite-Jacobi (CJ) relaxation to solve (5). Each CJ iteration updates the approximate solution (λ, μ), and consists of

$$\lambda_i = 1/\beta_i, \quad i = 1, \ldots, m$$

$$\gamma = B^T \lambda$$
$$\mu_i = 1/\gamma_j, \quad j = 1, \ldots, n$$
$$\beta = B\mu.$$

5 Optimal scaling and an instructive example

The scaling factors R and S obtained by the various scaling techniques described above can be interpreted as (approximate) solutions to various optimization models. The optimality conditions of these models were invoked as a means of finding optimal or nearly-optimal scaling factors. Naturally, one would like to know what is the *best* scaling model to use. For instance, is it best to minimize variance of the nonzero elements of rows/columns of the matrix or is it best to minimize a measure of the deviation of all elements from 1.0?

In the case of solving a square system of equations, one can make the argument that scaling should seek to minimize the condition number of the matrix, as doing so would increase the accuracy of the results. The latter is the problem of optimal scaling described above in model S and is widely recognized as the true scaling problem in the literature pertaining to the solution of linear systems. Many of the scaling models and techniques used to scale linear optimization problems were originally designed to scale systems of linear equations. Some of the scaling techniques presented herein provably decrease the condition number of matrices with certain properties.

In the context of the simplex method, the condition number of a basis provides a good measure of how far the corner is from being flat, where a 'flat corner' is a corner with incident edges forming angles of nearly 180 degrees. For instance, if one perturbs A by an appropriate Gaussian random matrix, then it is unlikely that any basis of A has a poor condition number [16]. Such a perturbation would bound the angles in each corner away from being flat. Unfortunately, if one scales the constraint matrix and the condition numbers of individual bases are increased, then the angles in each of these corners of the polytope are made increasingly flat. Intuition and reason would imply that flattening the corners of the polytope is not desirable for the simplex algorithm, rather reasonable progress is made when the corners of the polytope are not too flat. Therefore, it is possible for scaling to degrade the performance of the simplex algorithm by increasing the condition number of individual bases that may be encountered by the algorithm. To demonstrate this possibility, the following example is offered:

$$\begin{array}{lrrrrr} \min & -w & -x & -5y & -0.75z & \\ \text{s.t.} & 999{,}999w & & +7.5y & & = 7.5 \\ & 9w & +2x & +2.5y & +4z & = 11 \\ & 4w & +3x & +1.5y & & = 10.5 \\ & & & & w, x, y, z & \geq 0. \end{array}$$

In this problem, the condition number of the basis (columns 2, 3, and 4) at the solution is 3.3. When scaled, using ten iterations of binormalization scaling, the basis at the solution has a condition number of 6.6×10^3. When scaled, using ten iterations of geometric mean scaling, the basis at the solution has a condition number of $3.48 \times$

10^3. Using ten iterations of arithmetic mean scaling leads to a solution basis with a condition number of 3.22×10^4. In fact, all scaling techniques studied herein lead to a relatively ill-conditioned solution basis. Furthermore, if element $(1, 1)$ of the matrix were larger than $999,999$, the conditioning of the basis at the solution would be worse than that of the current linear optimization problem. It should now be clear that poorly scaled elements in columns that might never enter the basis could affect the conditioning of a basis that is factored in the course of solving a linear optimization problem using the simplex algorithm.

It follows from the above discussion that a simple decrease in the condition number of the constraint matrix is not necessarily beneficial for linear optimization. When a linear optimization problem is scaled prior to the application of the simplex algorithm, the concern is minimization of the condition number of *every* feasible basis. Since every feasible basis is not known *a priori*, the condition number of every basis, regardless of feasibility, would need to be minimized to address this concern. It can be argued that current scaling techniques use surrogate models to approximate this objective. One of the main objectives of the experimentation in the next section is to assess the impact of these scaling techniques on the condition number of all bases encountered in the course of the simplex algorithm, when the latter is applied to a number of linear optimization test problems.

6 Computational results

6.1 Problems and algorithms considered

In an effort to understand the difficulty involved in scaling linear optimization problems and to determine effective scaling methods or models, nearly a half billion executions of various linear optimization solvers were performed in the course of this study, using CPLEX, MINOS and GLPK. Most of the calculations were performed with CPLEX (version 12.2) [15] and were distributed across 22 processors. In all cases, the problems were scaled before they were passed to CPLEX for solution and CPLEX's scaling routine was turned off. In addition to CPLEX, MINOS (version 5.51) [20] and GLPK (Library Package 4.26) [19] were used to verify the results exhibited by CPLEX. The results of these runs are compiled in this section.

First, each scaling method and model was used in the scaling of 85 Netlib and 16 Kennington linear optimization problems. Table 1 provides a complete list of the problems solved, along with information on their size and number of nonzeros. In the table, a † denotes that this problem is considered part of the Kennington library, while all other problems are considered part of the Netlib library. A ⋄ denotes that this problem was randomly selected among manageably small problems to participate in the condition number study that is subsequently presented. Lastly, a ‡ denotes that this problem is poorly scaled. This final demarcation is described in more detail later in this section. There were several problems that were excluded from the condition number study because of the computational effort required to compute the condition number of these bases at every single iteration. For instance, it was not feasible to compute the condition number of every basis for problems with more than 100,000

Table 1 Netlib and Kennington problems solved with their dimensions and number of nonzeros

Problem	m	n	$\overline{Z}$	Problem	m	n	$\overline{Z}$
80bau3b ‡	2262	9799	21002	osa-14 † ‡	2337	52460	314760
adlittle ◇ ‡	56	97	383	osa-30 † ‡	4350	100024	600138
afiro ◇	27	32	83	osa-60 † ‡	10280	232966	1397793
agg2 ◇ ‡	516	302	4284	pds-02 †	2953	7535	16390
agg3 ◇ ‡	516	302	4300	pds-06 †	9881	28655	62524
agg ◇ ‡	488	163	2410	pds-10 †	16558	48763	106436
bandm ◇ ‡	305	472	2494	pds-20 †	33874	105728	230200
beaconfd ◇	173	262	3375	perold ◇ ‡	625	1376	6018
bnl1 ◇ ‡	643	1175	5121	pilot ‡	1441	3652	43167
bnl2 ‡	2324	3489	13999	pilot4 ◇ ‡	410	1000	5141
boeing1 ◇ ‡	351	381	3485	pilot87 ‡	2030	4883	73152
boeing2 ◇ ‡	166	143	1196	pilotnov ‡	975	2172	13057
bore3d ◇ ‡	233	315	1429	recipe ◇ ‡	91	180	663
brandy ◇ ‡	220	249	2148	sc105 ◇	105	103	280
capri ◇ ‡	271	353	1767	sc205 ◇	205	203	551
cre-a † ‡	3516	4067	14987	sc50a ◇	50	48	130
cre-b † ‡	9648	72447	256095	sc50b ◇	50	48	118
cre-c † ‡	3068	3678	13244	scagr25 ◇	427	500	1554
cre-d † ‡	8926	69980	242646	scagr7 ◇ ‡	129	140	420
cycle ◇ ‡	1903	2857	20720	scfxm1 ◇ ‡	330	457	2589
czprob ◇ ‡	929	3523	10669	scfxm2 ◇ ‡	660	914	5183
d2q06c ‡	2171	5167	32417	scfxm3 ◇ ‡	990	1371	7777
d6cube ‡	415	6184	37704	scorpion◇	388	358	1426
degen2 ◇	444	534	3978	scrs8 ◇	490	1169	3182
degen3	1503	1818	24646	scsd1 ◇	77	760	2388
e226 ◇ ‡	223	282	2578	scsd6 ◇	147	1350	4316
etamacro ◇ ‡	400	688	2409	scsd8 ◇	397	2750	8584
fffff800 ◇ ‡	524	854	6227	sctap1 ◇	300	480	1692
finnis ◇ ‡	497	614	2310	sctap2 ◇	1090	1880	6714
fit1d ◇ ‡	24	1026	13404	sctap3 ◇	1480	2480	8874
fit1p ◇ ‡	627	1677	9868	seba ‡	515	1028	4352
fit2d ‡	25	10500	129018	share1b ◇ ‡	117	225	1151
fit2p ‡	3000	13525	50284	share2b ◇ ‡	96	79	694
ganges ◇	1309	1681	6912	shell ◇	536	1775	3556
greenbea	2392	5405	30877	ship04l ◇	402	2118	6332
greenbeb	2392	5405	30877	ship04s ◇	402	1458	4352
grow15 ◇	300	645	5620	ship08l ◇	778	4283	12802
grow22 ◇	440	946	8252	ship08s ◇	779	2387	7114
grow7 ◇	140	301	2612	ship12l ◇	1151	5427	16170
israel ◇ ‡	174	142	2269	ship12s ◇	1151	2763	8179
kb2 ◇ ‡	43	41	286	stair ◇	356	467	3856

Table 1 (*Continued*)

Problem	m	n	$\overline{Z}$	Problem	m	n	$\overline{Z}$
ken-07 $^{\diamond\,\dagger}$	2426	3602	8404	standata$^{\diamond\,\ddagger}$	359	1075	3031
ken-11 †	14694	21349	49058	standgub$^{\diamond\,\ddagger}$	361	1184	3139
ken-13 †	28632	42659	97246	standmps$^{\diamond\,\ddagger}$	467	1075	3679
ken-18 †	105127	154699	358171	stocfor1$^{\diamond\,\ddagger}$	117	111	447
lotfi $^{\diamond\,\ddagger}$	153	308	1078	stocfor2$^{\diamond\,\ddagger}$	2157	2031	8343
maros $^{\diamond\,\ddagger}$	846	1443	9614	tuff $^{\diamond\,\ddagger}$	333	587	4520
maros-r7	3136	9408	144848	vtp-base$^{\diamond\,\ddagger}$	198	203	908
modszk1 $^{\diamond}$	687	1620	3168	wood1p $^{\diamond\,\ddagger}$	244	2594	70215
nesm ‡	662	2923	13288	woodw ‡	1098	8405	37474
osa-07 $^{\dagger\,\ddagger}$	1118	23949	143694				

nonzeros. CPLEX was used to compute the exact condition number at every iteration using CPX_EXACT_KAPPA.

For each of these 101 test problems, each scaling method and model was tested using one, two, four, six, and eight iterations, except equilibration, which only requires a single iteration. Table 2 depicts the 51 different applications of scaling, which were applied to the 101 different test problems. In addition, the 101 problems were tested without scaling. The condition number of the basis and the elapsed solution time were stored at every iteration of the primal and dual simplex algorithms. The solution time in all subsequent figures is the time it takes to solve the optimization problem, excluding the time it takes to scale the problem. In addition, the scaling was applied with and without presolve. In all, there were

$$\begin{aligned}&(N_{\text{sol}}) \times (52 \text{ scaling applications}) \times (N_{\text{pre}}) \times (N_{\text{pro}})\\ &\quad + (N_{\text{sol}}) \times (2350 \text{ scaling applications}) \times (N_{\text{pre}}) \times (N_{\text{pro}})\\ &\quad = 1{,}940{,}816 \text{ distinct experiments},\end{aligned}$$

where N_{sol} equals four and represents the number of different simplex solution techniques used (CPLEX's primal and dual simplex, GLPK, and MINOS), N_{pre} equals two and represents the executions required to toggle presolve for each solution technique, and N_{pro} equals 101 and represents the number of Netlib and Kennington problems solved. In the pursuit of accurate solution and scaling times, each distinct experiment was performed five times, and the median time was selected as the representative time required. In order to accurately track the elapsed solution time and condition number at every iteration, CPLEX was stopped at every iteration, and the necessary data was calculated and stored. This resulted in an explosive growth of processing time and I/O, which led to nearly half a billion CPLEX executions.

The reasons for using MINOS and GLPK in addition to CPLEX were three-fold. First, as previously mentioned, they were used to verify the results obtained using CPLEX. Second, they were used to provide access to an open-source code, so the reader would be aware of any additional actions that were taking place in the code that might affect the scaled problem. Lastly, each solver uses different pricing techniques.

Table 2 The scaling methods/models used and the associated iteration counts

k	Method/Model	Iter.	*k*	Method/Model	Iter.	*k*	Method/Model	Iter.
1	No Scaling	0	**18**	Entropy	2	**36**	L1-norm	6
2	Arithmetic Mean	1	**19**	Entropy	4	**37**	L1-norm	8
3	Arithmetic Mean	2	**20**	Entropy	6	**38**	L2-norm	1
4	Arithmetic Mean	4	**21**	Entropy	8	**39**	L2-norm	2
5	Arithmetic Mean	6	**22**	Equilibration	1	**40**	L2-norm	4
6	Arithmetic Mean	8	**23**	Geometric Mean	1	**41**	L2-norm	6
7	deBuchet $p = 1$	1	**24**	Geometric Mean	2	**42**	L2-norm	8
8	deBuchet $p = 1$	2	**25**	Geometric Mean	4	**43**	Linf-norm	1
9	deBuchet $p = 1$	4	**26**	Geometric Mean	6	**44**	Linf-norm	2
10	deBuchet $p = 1$	6	**27**	Geometric Mean	8	**45**	Linf-norm	4
11	deBuchet $p = 1$	8	**28**	IBM MPSX	1	**46**	Linf-norm	6
12	deBuchet $p = 2$	1	**29**	IBM MPSX	2	**47**	Linf-norm	8
13	deBuchet $p = 2$	2	**30**	IBM MPSX	4	**48**	Binormalization	1
14	deBuchet $p = 2$	4	**31**	IBM MPSX	6	**49**	Binormalization	2
15	deBuchet $p = 2$	6	**32**	IBM MPSX	8	**50**	Binormalization	4
16	deBuchet $p = 2$	8	**33**	L1-norm	1	**51**	Binormalization	6
17	Entropy	1	**34**	L1-norm	2	**52**	Binormalization	8
			35	L1-norm	4			

For instance, MINOS uses a sectional variant of Dantzig's rule [20]. CPLEX utilizes steepest-edge and Devex rules [3]. The GLPK package is capable of utilizing several different pricing techniques, but its default is set to steepest-edge.

One might consider that, having tested a scaling algorithm using eight iterations, it is unnecessary to test the same algorithm with fewer iterations. This thought process is incorrect. Consider a two-by-two matrix containing four positive nonzeros with large diagonal elements. Now, consider the effect of scaling all nonzero elements to order unity using any of the scaling algorithms described above. As the system is progressively scaled with each iteration, the system approaches singularity. Therefore, fewer iterations are preferred in such an instance.

6.2 Measures of comparison

Four measures were used to evaluate the quality of each scaling technique: scaling time, solution time, solution iterations, and maximum condition number. Scaling time is a necessary measure of quality because a scaling technique that requires too much time (in relation to, or as a fraction of the possible reduction in solution time) is unsuccessful. The solution time and the required number of iterations are integrally related. If a scaling technique can succeed in reducing the required number of simplex iterations, then it is likely the scaling technique has contributed to a decrease in the required solution time. If these objectives can be achieved, then the scaling technique has contributed to the overall success of the solver. Lastly, the maximum condition number of the bases encountered during solution is an appropriate measure because of its relationship to accuracy and solution time. The log of the condition number to the base ten is an upper bound on the number of decimal places lost. That is, if the condition number were 10^{16}, then up to 16 digits of precision may be lost

(or all 16 decimal places available in standard double-precision floating-point). For more information on the effect of the condition number on the solution accuracy, the reader is referred to the section "Measuring problem sensitivity with basis condition number" in [15]. The condition number could be periodically evaluated or approximated, and suitable numerical methods, such as refactorization or interval arithmetic, could be invoked to ensure the required accuracy of the results, albeit at the increase of computational time.

For any chosen method of comparison, it is necessary to rely on one or more indicators that accurately reflect the success or failure of any scaling technique over a collection of test problems. One such indicator is how well the technique performs on average with respect to the aforementioned measures. Another indicator of success is the standard deviation. A good scaling technique is one with *predictable results* either for all problems or for a group of problems sharing a specific property. If a scaling technique is the 'best' amongst all techniques studied for half of all problems but the 'worst' for the other half, then it may be recognized as a 'good' scaling technique if a uniform property is recognized for the former half. However, if no such property is readily recognizable, then the scaling technique should be considered erratic and undesirable. Without predictable results, the consumer of the software may be unwilling to assume the risks of scaling. These risks might include poor solution accuracy and poor solution time.

Since the magnitude of the performance metrics (simplex iterations, solution time, and maximum condition number) depend on the nature and size of the problem, it is necessary to normalize the results. For example, consider simplex iterations and follow Example 1. For each linear optimization problem, determine the scaling technique requiring the most simplex iterations and divide the simplex iterations required by all other scaling techniques by this maximum. Each problem will now contain normalized simplex iterations, e.g., Table 1. To compare the average performance of the various scaling techniques, these normalized results are averaged and their standard deviations are computed. The average is indicative of the average performance of a scaling application over the 101 problems, and the standard deviation illustrates the variation in performance. Error bars in each of the figures described below are indicative of one standard deviation from the mean. Example 1 is provided below to clarify the explanation of this normalization procedure.

Example 1 Assume the data returned by the linear optimization code is as follows:

	Iterations required		
	Scaling technique 1	Scaling technique 2	Scaling technique 3
Problem A	1500	1450	1700
Problem B	30	32	28
Problem C	400	200	500

This example assumes that only three problems were tested and only three scaling techniques were analyzed. First, normalize the data across each problem:

	Iterations required		
	Scaling technique 1	Scaling technique 2	Scaling technique 3
Problem A	$1500/1700 = 0.88$	$1450/1700 = 0.85$	$1700/1700 = 1.0$
Problem B	$30/32 = 0.94$	$32/32 = 1.0$	$28/32 = 0.88$
Problem C	$400/500 = 0.8$	$200/500 = 0.4$	$500/500 = 1.0$

Then average the normalized numbers and take their standard deviation:

$$\text{Scaling technique } 1 = 0.87 \pm 0.069$$

$$\text{Scaling technique } 2 = 0.75 \pm 0.313$$

$$\text{Scaling technique } 3 = 0.96 \pm 0.072$$

The "ratio-to-worst" value is used rather than the more typical "ratio-to-best" performance metric because there exist singular outliers that adversely affect the latter.

6.3 Results and discussion

For the sake of brevity, we aggregate the data across the Netlib and Kennington data sets. The use of a presolve routine had no discernible effect on the relative success or failure of a given a scaling technique. The experiments that were performed with CPLEX utilized its primal as well as dual simplex algorithm. The results for the primal simplex algorithm are contained in the online resource, while the results for the dual simplex algorithm are presented in this subsection. In addition, a series of experiments investigating the performance of combinations of scaling techniques was performed on selected problems. The results of these experiments are all contained in the online resource, as they do not differ from the results described in this section.

The first series of experiments were designed to study the effect of individual scaling methods. These results are presented in Figs. 1–8. The x-axis of these figures represents the 52 different scalings normalized across the 101 different problems, with no scaling and equilibration scaling highlighted in gray with a $*$ rather than a $\bullet$ denoting the mean. In Table 2, the bolded numbers correspond to the values on the x-axis of these figures. Each number indicates the scaling method/model and the associated iteration count.

In Fig. 1, the normalized scaling time for the presolved problem is displayed. As seen in this figure, scaling techniques with non-trivial termination criteria require a greater amount of CPU time, and their CPU time varies more than those techniques that have no termination criteria or relatively simple criteria. The proportion of scaling time to the overall solution time will vary depending on the problem but should remain relatively small in a well-designed code. That is, the normalized total solution time (including the time to scale) does not significantly differ from the normalized solution times (excluding the time to scale).

Figures 2, 3, and 4 display the normalized solution times exhibited by CPLEX, MINOS, and GLPK, respectively, for each scaling technique. Some interesting points are illustrated in these figures. A low variation in solution time should be recognized

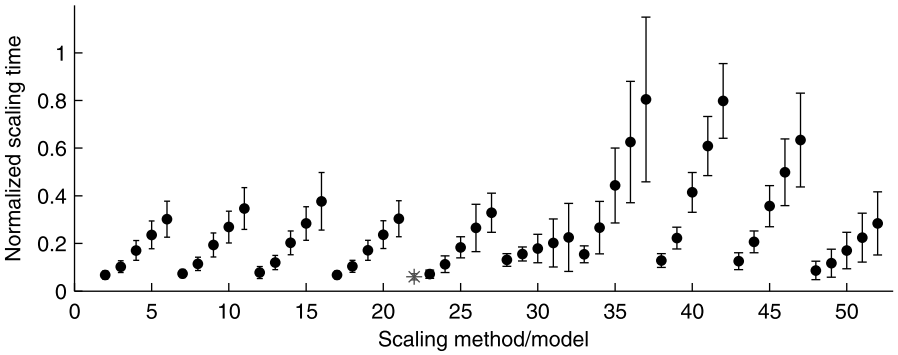

Fig. 1 The normalized time to scale

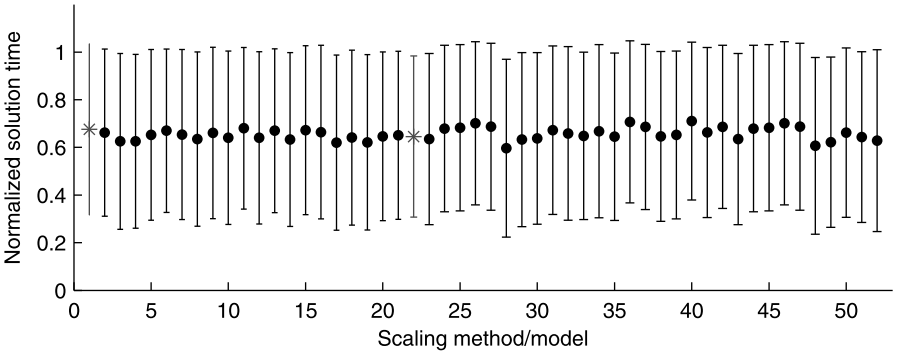

Fig. 2 The normalized solution time with dual simplex using CPLEX

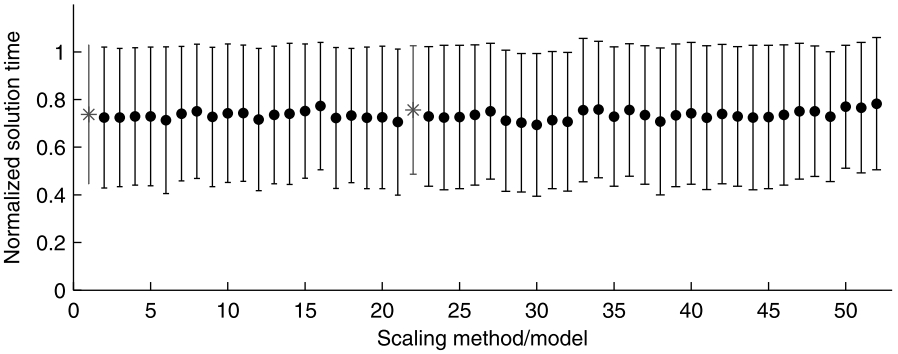

Fig. 3 The normalized solution time using GLPK

as a desired property of a good scaling algorithm. That is, it is desirable to have a scaling algorithm that does not woefully underperform and then incredibly outperform other algorithms. The figures indicate that there exist no clear "best" scaling

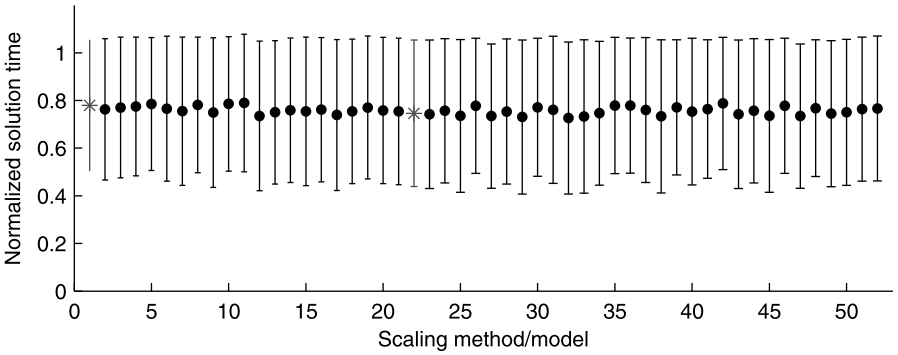

Fig. 4 The normalized solution time using MINOS

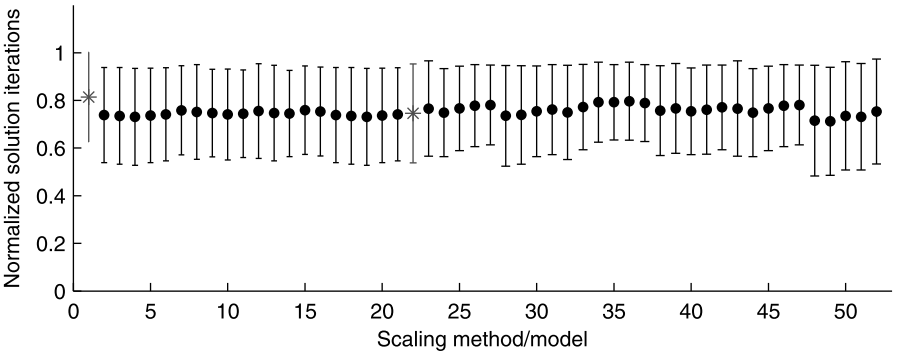

Fig. 5 The normalized solution iterations with dual simplex using CPLEX

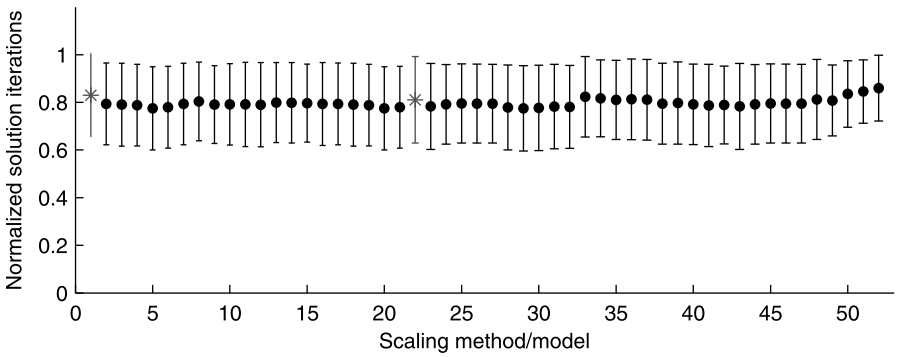

Fig. 6 The normalized solution iterations using GLPK

technique. Within the standard deviation, the absence of scaling appears to perform on a par with each scaling technique. At the same time, the performance of some scaling techniques on individual problems seems to indicate that the design of an in-

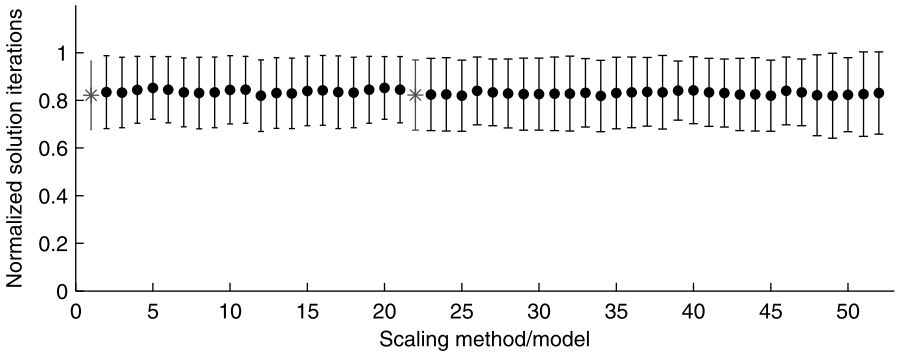

Fig. 7 The normalized solution iterations using MINOS

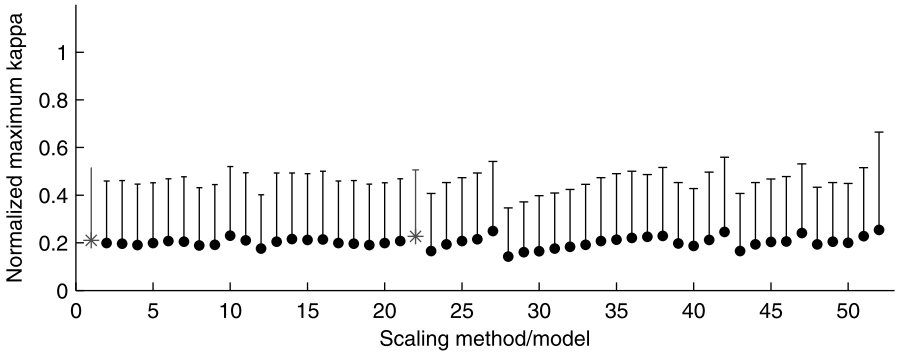

Fig. 8 The normalized maximum condition number with dual simplex using CPLEX

telligent scaling algorithm could lead to a lower solution time. The question remains open as to whether such an algorithm exists and possesses a competitive scaling time.

Figures 5–7 depict the normalized solution iterations for each solver and scaling technique. These figures tell a similar story to those of the normalized solution time. Again, a desired property of a good scaling algorithm is to avoid wild variations in the number of solution iterations relative to other scaling algorithms.

The normalized maximum condition numbers of the bases encountered by CPLEX during dual simplex iterations are depicted in Fig. 8. The vertical bars in the figure represent one standard deviation from the mean maximum condition number. The bars in the negative direction (i.e., the mean minus one standard deviation) are not shown in the figure, because the mean maximum condition number and the worst-case maximum condition number are our primary concerns. The worst-case condition number possibly plays a role in the poor solution times exhibited by some of these algorithms.

In terms of the worst-case maximum condition number measure, IBM MPSX scaling performs fairly well compared to the other scaling techniques. In Fig. 8, notice that scaling algorithms 28–31 have relatively low worst-case maximum condition numbers relative to other scaling techniques. Therefore, this scaling technique should

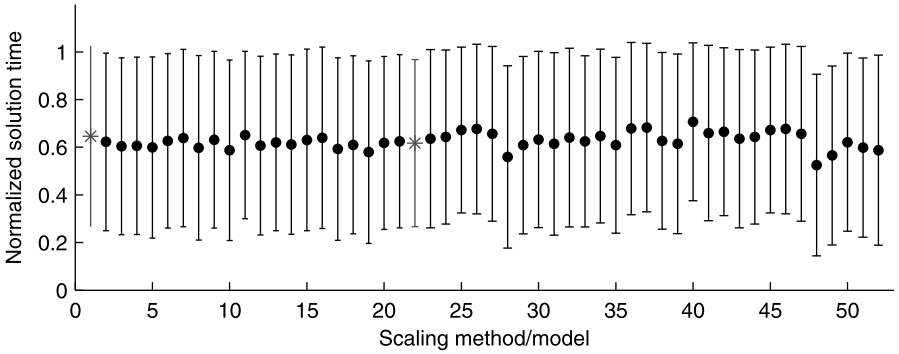

Fig. 9 The normalized solution time with dual simplex using CPLEX

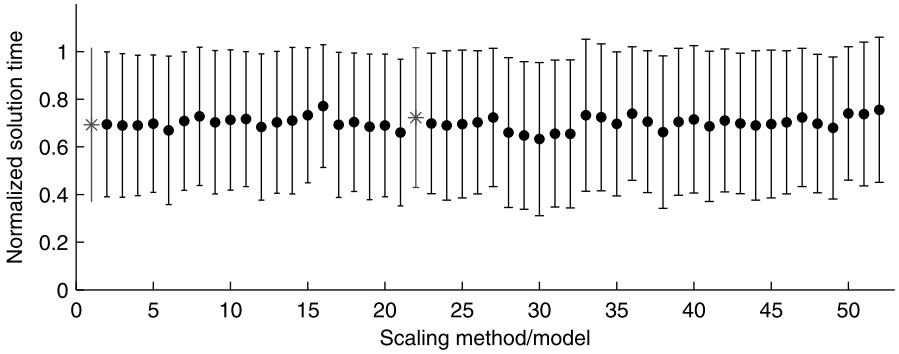

Fig. 10 The normalized solution time using GLPK

be recognized as a potentially suitable scaling technique for use in linear optimization codes. Most importantly, Fig. 8 confirms that there is a diminishing return to scaling as more iterations are applied. Some scaling techniques exhibit no discernable advantage to additional scaling iterations, and some techniques even exhibit a higher maximum condition number as more scaling iterations are applied.

The next series of experiments recognizes the fact that some problems in the Netlib and Kennington libraries are inherently well-scaled. As it is impractical to compute the condition number of all possible bases, it is common in linear optimization to consider a matrix well-scaled if its elements are of similar magnitude. A ‡ is used in Table 1 to denote problems that are poorly scaled. For this study, any matrix with an initial variance above four is considered poorly scaled. The number four was chosen because it was used as the termination criterion in this paper for IBM MPSX scaling. Other researchers have recommended a termination criterion as high as ten [2]. Figures 9–14 correspond to Figs. 2–7 with the only difference being that well-scaled problems have been removed from consideration. As a result, about 30% of the original problems were removed. As seen in the figures, there is no discernible difference in the results after removal of the well-scaled problems.

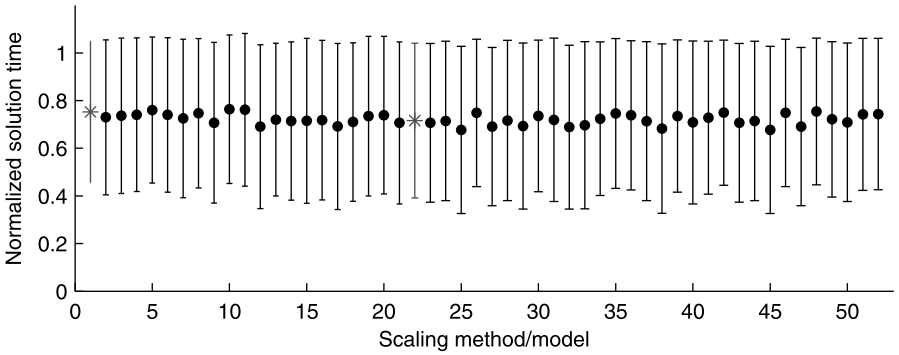

Fig. 11 The normalized solution time using MINOS

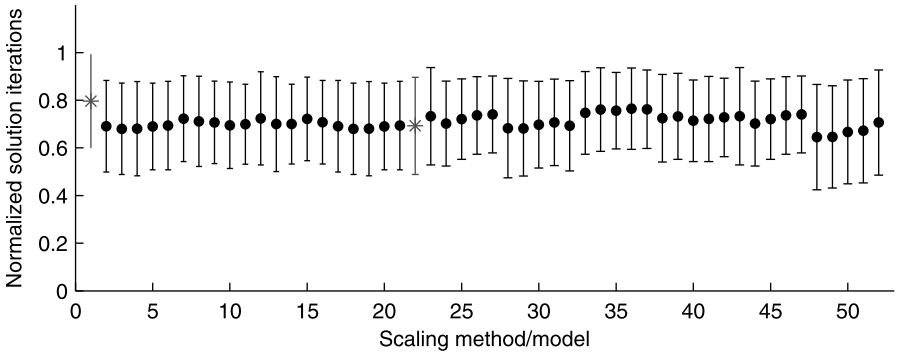

Fig. 12 The normalized solution iterations with dual simplex using CPLEX

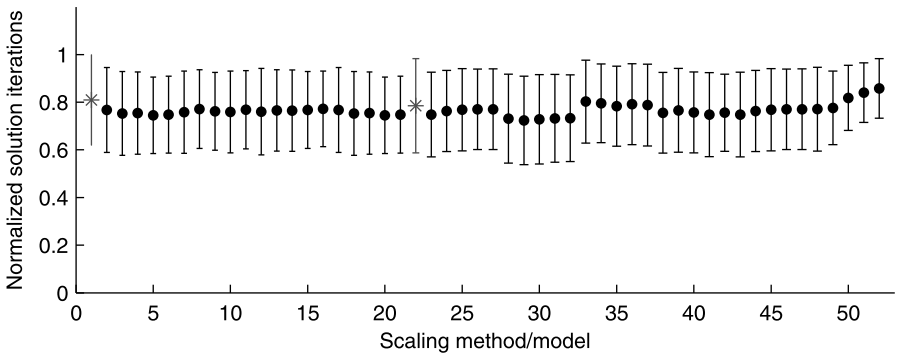

Fig. 13 The normalized solution iterations using GLPK

It is interesting to note that the normalized solution iterations in Fig. 12 appear to increase when no scaling technique is used relative to employing a scaling technique. The same observation can be made for CPLEX's primal simplex algorithm (the

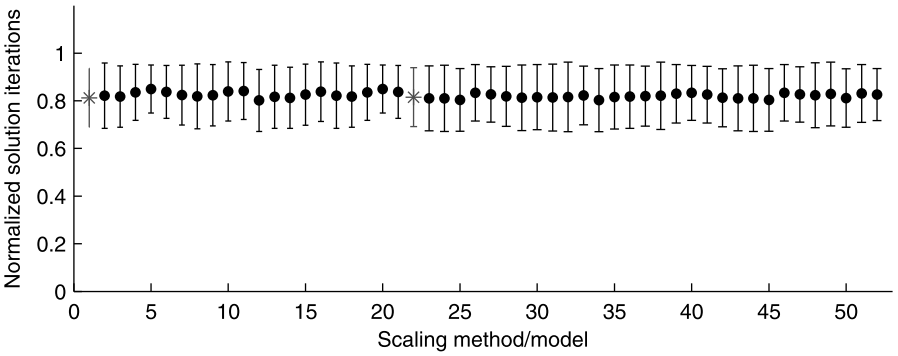

Fig. 14 The normalized solution iterations using MINOS

results are provided in the online resource). However, GLPK and MINOS are not affected as much by the absence of scaling as seen in Figs. 6 and 7. In the absence of scaling, tolerances play a more important role in determining the performance of the overall algorithm. The use of absolute tolerances is significantly more efficient than the use of dynamic tolerances, which suggests that CPLEX utilizes more absolute tolerances than GLPK and MINOS. The utilization of more absolute tolerances would allow CPLEX to solve the problems more rapidly, but also require that the problems be well-scaled. In the event that problems are not well-scaled, performance will suffer in the absence of scaling. A more extensive use of absolute tolerances would certainly explain why CPLEX performs equilibration scaling by default.

The last series of experiments investigated the use of combinations of the scaling techniques described above. In these experiments, there are ten different scaling techniques that are tested across one, two, four, six, eight iterations, and equilibration. There are 2350 different combinations of these scaling techniques and models. Naturally, the combinations of scaling techniques are going to exhibit a higher overall scaling time than the single scaling techniques exhibited in the previous figures. A corresponding reduction in solution time would be required to warrant the use of these combinations. The results from these computations are reported in the online resource and, once again, indicate no real trend or pattern in solution time relative to the combinations of scaling techniques employed here. The same lack of pattern is evident in the data collected on the normalized solution iterations.

Lastly, for the normalized maximum condition number of the bases with various solution algorithms, the results in the online resource show, once again, the same result as the figures for a single scaling technique (Fig. 8). That is, IBM MPSX scaling is relatively efficient at controlling the maximum condition number of the bases. However, its combination with some scaling applications proved to degrade its performance relative to this measure.

7 Conclusions

This study should make the reader appreciate the complexity of the scaling problem as it relates to linear optimization. Simply applying techniques that work well in

practice for the solution of linear systems (i.e., for solving $Ax = b$, where A is an $n \times n$ matrix and b is a column vector) is *not* a solution to the underlying problem of scaling a linear optimization problem, as illustrated in Sect. 5. It should also be apparent from the results of Sect. 6 that a viable solution to the scaling problem does not just perform well on average, but has a satisfactory worst-case as measured by the standard deviation of solution iterations, solution time, and maximum condition number.

Despite the potential increase in the condition number of a basis or a series of bases, it is difficult not to recommend performing equilibration scaling to at least provide a point of reference for absolute tolerances. It is disconcerting that if a scaling technique causes a problem to become poorly scaled, then certain linear optimization solvers tend to undertake a greater number of iterations relative to the absence of scaling. Further evidence of this phenomenon is provided in the online resource, where CPLEX's primal simplex algorithm is studied. In addition, too many iterations of the same scaling technique (e.g., geometric mean) can adversely affect a linear optimization problem. Presolve followed by equilibration scaling leads to condition numbers and simplex solution times and iterations that are competitive with those observed using other scaling techniques.

Acknowledgements We are thankful to three reviewers, whose insights and generous comments helped us improve the presentation of this material significantly.

References

1. Bauer, F.L.: Optimally scaled matrices. Numer. Math. **5**, 73–87 (1963)
2. Benichou, M., Gauthier, J.M., Hentges, G., Ribiere, G.: The efficient solution of large-scale linear programming problems—Some algorithmic techniques and computational results. Math. Program. **13**, 280–322 (1977)
3. Bixby, R.E.: Solving real-world linear programs: A decade and more of progress. Oper. Res. **50**, 3–15 (2002)
4. Bradley, A.: Algorithms for the equilibration of matrices and their application to limited-memory quasi-Newton methods. PhD thesis, Stanford University, Stanford, California (2010)
5. Chvátal, V.: Linear Programming. W.H. Freeman, New York (1983)
6. Curtis, A.R., Reid, J.K.: On the automatic scaling of matrices for Gaussian elimination. J. Inst. Math. Appl. **10**, 118–124 (1972)
7. Dahlquist, G., Björck, Å.: Numerical Methods. Prentice Hall, Englewood Cliffs (1963)
8. de Buchet, J.: Experiments and statistical data on the solving of large-scale linear programs. In: Hertz, D.A., Melese, J. (eds.) Proceedings of the Fourth International Conference on Operational Research, pp. 3–13. Wiley-Interscience, New York (1966)
9. Elble, J.: Scaling linear programs: A comprehensive case study. Master's thesis, University of Illinois Urbana-Champaign, Urbana, IL (2007)
10. Elble, J.M., Sahinidis, N.V.: Matrix binormalization on a GPU. In: Lecture Notes in Computer Science (2009, accepted)
11. Forsythe, G.E., Straus, E.G.: On best conditioned matrices. Proc. Am. Math. Soc. **6**, 340–345 (1955)
12. Fulkerson, D.R., Wolfe, P.: An algorithm for scaling matrices. SIAM Rev. **4**, 142–146 (1962)
13. Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins University Press, Baltimore (1996)
14. Hamming, R.W.: Introduction to Applied Numerical Analysis. McGraw-Hill, New York (1971)
15. IBM: IBM ILOG CPLEX Optimization Studio, Version 12.2 User's Manual, Armonk, NY (2010)
16. Kelner, J.A., Spielman, D.A.: A randomized polynomial-time simplex algorithm for linear programming (Preliminary version). Electron. Colloq. Comput. Complex. **156**, 1–17 (2005)

17. Larsson, T.: On scaling linear programs—Some experimental results. Optimization **27**, 335–373 (1993)
18. Livne, O.E., Golub, G.H.: Scaling by binormalization. Numer. Algorithms **35**, 97–120 (2004)
19. Makhorin, A.: GLPK—GNU linear programming kit. http://www.gnu.org/software/glpk/glpk.html (2008)
20. Murtagh, B.A., Saunders, M.A.: MINOS 5.5 user's guide. Technical report, Department of Operations Research, Stanford University, Stanford, CA (1998)
21. Orchard-Hays, W.: Advanced Linear Programming Computing Techniques. McGraw-Hill, New York (1968)
22. Pierre, D.A.: An optimal scaling method. IEEE Trans. Syst. Man Cybern. **SMC-17**, 2–6 (1987)
23. Rothblum, U.G., Schneider, H.: Characterizations of optimal scalings of matrices. Math. Program. **19**, 121–136 (1980)
24. Tomlin, J.A.: On scaling linear programming problems. Math. Program. Stud. **4**, 146–166 (1975)
25. van der Sluis, A.: Condition numbers and equilibration of matrices. Numer. Math. **14**, 14–23 (1969)
26. van der Sluis, A.: Condition, equilibration and pivoting in linear algebraic systems. Numer. Math. **15**, 74–86 (1970)
27. von Golitschek, M.: An algorithm for scaling matrices and computing the minimum cycle mean in a digraph. Numer. Math. **35**, 45–55 (1980)