A Regularized Jacobi Method for Large-Scale Linear Programming

Kallio, M.J., Ruszczynski, A. and Salo, S.

IIASA Working Paper
WP-93-061

November 1993
A regularized Jacobi method for large-scale linear programming

Markku Kallio
Andrzej Ruszczyński
Seppo Salo

WP-93-61
November 1993
A regularized Jacobi method for large-scale linear programming

Markku Kallio
Andrzej Ruszczcyński
Seppo Salo

WP-93-61
November 1993
Abstract

A parallel algorithm based on Jacobi iterations is proposed to minimize the augmented Lagrangian functions of the multiplier method for large-scale linear programming. Sparsity is efficiently exploited for determining stepsizes (column-wise) for the Jacobi iterations. Linear convergence is shown with convergence ratio depending on sparsity but not on the penalty parameter and on problem size. Employing simulation of parallel computations, an experimental code is tested extensively on 68 Netlib problems. Results are compared with the simplex method, an interior point algorithm and a Gauss-Seidel approach. We observe that speedup against the simplex method generally increases with the problem size, while the parallel solution times increase slowly, if at all. Our preliminary results compared with the other two methods are highly encouraging as well.

Keywords: Large-Scale Linear Programming, Augmented Lagrangians, Parallel Computing, Partial Equilibrium.
1. Introduction

The main objective of this paper is to develop a parallel procedure for solving the standard linear programming problem

$$\min c^T x$$

$$Ax = b$$

$$l \leq x \leq u,$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $c$, $l$ and $u$ are some vectors in $\mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A$ is an $m \times n$ matrix. We think of $A$ as a large-scale and sparse matrix, which does not necessarily possess other structural properties. To begin, we shall consider the multiplier method [12, 22] for solving (1.1). Using Jacobi-type iterations (aimed at parallel computing), we develop a simple solution algorithm for augmented Lagrangian subproblems. It combines and develops the ideas of our earlier theoretical results reported in [24] and of the tatonnement procedure of [14] based on Gauss-Seidel iterations.

After some preliminary concepts and results we discuss weak convergence of Jacobi iterations in Section 4. We introduce a safe stepsize for the iterations to ensure convergence. Sparsity will be efficiently exploited for determining such stepsizes separately for each column and for each iteration. We indicate experimentally that such stepsizes indeed are reasonably large. In Section 5, convergence at a linear rate is shown with convergence ratio depending on sparsity. It is interesting to note, however, that the size of the problem and the penalty parameter do not appear in the rate of convergence estimates. Both for the safe stepsize and for the rate of convergence much stronger results are presented for linear programs as compared with more general results for convex programming developed in [24].

The code of [14] for Gauss-Seidel iterations was revised for Jacobi iterations resulting in an experimental code called Jacobi. This implementation is described in Section 6. Particular attention is paid to the definition of primal and dual tolerances, as well as to the penalty parameters: dual tolerances are defined by column, and primal tolerances as well as the penalty parameters are defined by row. Each one of them is recalculated in each iteration. For these reasons, scaling of (1.1) appears irrelevant for the code: Jacobi is practically invariant to scaling.

Extensive computational tests on 68 Netlib [19] problems are reported in Section 7. We simulate parallel computations on an HP9000/720 serial computer. In such simulations, the communication time for parallel processors is omitted while estimating the parallel run times based on serial run times.

First, a basic version of Jacobi is tested against the simplex method employing Minos 5.1 [18]. We observe that speedup against the simplex method generally increases with the problem size, while the parallel solution time increases only moderately. Three additional test runs are reported on larger problems than those above. For these problems with 5 to 16 thousand rows, the parallel run time appeared to be about the same or smaller than the average for the 68 smaller problems. However, the run time is sensitive to the precision requirement: decreasing the tolerances by a factor of ten
increases the average parallel run time for the 68 Netlib problems from 0.18 sec to 0.85 sec.

Variants of Jacobi are tested in Section 7.2. We consider the safe stepsize against a constant stepsize, as well as possible line search procedures for accelerating the Jacobi iterations. Sensitivity testing with respect to the program parameters is reported in Section 7.3.

Finally in Section 7.4, a comparison of Jacobi is presented against serial algorithms based on Gauss-Seidel iterations (see [14]) and on an interior point algorithm [1]. In applications where extreme precision is not required for the solution, e.g. for economic models, our preliminary results compared with the simplex method, Gauss-Seidel and interior point methods are highly encouraging.

2. Preliminaries

The use of the duality theory for solving linear programs is almost as old as the theory of linear programming itself (see [5] and the references therein). Nowadays, the most efficient linear programming algorithms, both simplex-like and interior-point methods, heavily exploit duality. Duality is also the main theoretical foundation of decomposition approaches to large structured problems, like, e.g., the decomposition principle of Dantzig and Wolfe [6]. The aim of this paper is to use duality in a non-standard way to develop a parallel computational method for (1.1) under the assumption that the constraint matrix A is sparse but does not necessarily possess any additional structural properties.

Let us recall that the (ordinary) Lagrangian for (1.1) can be defined as

\[ L(x,\pi) = c^T x + \pi^T (b - Ax), \]

where \( \pi \in \mathbb{R}^m \) is the vector of multipliers associated with the equality constraint \( Ax = b \). The Lagrangian can be used to define the dual function

\[ \hat{L}(\pi) = \inf_{l \leq x \leq u} L(x,\pi) = b^T \pi + \sum_{j=1}^{n} \hat{L}_j(\pi) \]

with

\[ \hat{L}_j(\pi) = \inf_{l_j \leq x_j \leq u_j} (c_j - \pi^T A_j) x_j, \]

where \( A_j \) denotes the jth column of \( A \).

The dual problem associated with (1.1) is formulated as

\[ \max_{\pi \in \mathbb{R}^m} \hat{L}(\pi). \]

Much is known about problem (2.3) and its relation to (1.1). For our purposes the most important facts are that \( \hat{L} \) is a concave piecewise linear function (which is readily seen from (2.2)) and that the following duality theorem holds.

**Proposition 1.** Assume that (1.1) has an optimal solution. Then (2.3) has an optimal solution and
(a) for every optimal solution \( \hat{x} \) of (1.1) and every optimal solution \( \hat{\pi} \) of (2.3)
\[
c^T \hat{x} = \hat{L}(\hat{\pi});
\]

(b) for every optimal solution \( \hat{\pi} \) of (2.3) a point \( l \leq \hat{x} \leq u \) is an optimal solution of
(1.1) if and only if
\[
(c_j - \hat{\pi}^T A_j) \hat{x}_j = \min_{l_j \leq \hat{x}_j \leq u_j} (c_j - \hat{\pi}^T A_j)x_j , \quad j = 1, \ldots, n
\]
\[
A \hat{x} = b.
\]

However, two main difficulties impede the straightforward application of the Lagrangian duality in linear programming. First, the dual (2.3) is a nonsmooth optimization problem with a concave piecewise-linear objective. Since we are still in the class of linear programming problems, it cannot be substantially easier than the solution of (1.1), unless the problem has a special structure. Secondly, not every point \( x \) defined by (2.4) satisfies (2.5); auxiliary procedures are necessary to identify the primal solution. There are many ways of overcoming these difficulties. One is to generate a sequence of primal-dual pairs in such a way that it always satisfies one of conditions (2.4)-(2.5) and successively decreases the violation of the other. Another possibility, that we are going to follow here, is to embed the original linear programming problem into a family of nonlinear problems which have some desirable properties of the dual function and of the set of Lagrangian minimizers. As an example of such an approach may serve the primal-dual logarithmic barrier method in which Lagrangian minimizers are unique and the system of primal and dual feasibility conditions can be solved by Newton’s method.

We are going to base our work on yet another idea, namely on the augmented Lagrangian duality. The augmented Lagrangian for (1.1) can be defined as follows:
\[
\Lambda(x, \pi) = c^T x + \pi^T (b - Ax) + \frac{1}{2}\rho \|b - Ax\|^2,
\]
where \( \rho \) is a positive penalty parameter. Similarly to (2.1)-(2.3) we can introduce the regularized dual function
\[
\hat{\Lambda}(\pi) = \inf_{l \leq x \leq u} \Lambda(x, \pi),
\]
the set of minimizers
\[
\hat{X}(\pi) = \{l \leq x \leq u : \Lambda(x, \pi) = \hat{\Lambda}(\pi)\}
\]
and the regularized dual problem
\[
\max_{\pi \in \mathbb{R}^m} \hat{\Lambda}(\pi).
\]

Referring to [25], we note that problem (2.7) is equivalent to a partial equilibrium problem, where rows of \( A \) refer to commodities and quantity units for these commodities are chosen so that the slope of each one of the (linear) price functions is \(-\rho\). In fact, our earlier work [14] originated from a Gauss-Seidel approach to a partial equilibrium model for the world forest sector [13].

For the augmented Lagrangian we can state a theorem similar to Proposition 1 (see, e.g., [23, 2]).
**Proposition 2.** Assume that (1.1) has an optimal solution. Then (2.9) has an optimal solution and

(a) for every optimal solution $\hat{x}$ of (1.1) and every optimal solution $\hat{\pi}$ of (2.9)

$$c^T \hat{x} = \hat{A}(\hat{\pi});$$

(b) for every optimal solution $\hat{\pi}$ of (2.9) a point $l \leq \hat{x} \leq u$ is an optimal solution of (1.1) if and only if

$$\Lambda(\hat{x}, \hat{\pi}) = \min_{l \leq x \leq u} \Lambda(x, \hat{\pi}). \quad (2.10)$$

Let $A_j$ denote the $j$th column of $A$. We define for $j = 1, \ldots, n$ the functions

$$\tilde{A}_j(x_j, \hat{x}, \pi) = (c_j - \pi^T A_j) x_j + \frac{1}{2} \rho ||b - A_j x_j - \sum_{s \neq j} A_s \hat{x}_s||^2. \quad (2.11)$$

Then (2.10) can be rewritten component-wise as

$$\tilde{A}_j(\hat{x}_j, \hat{x}, \pi) = \min_{l \leq x_j \leq u_j} \tilde{A}_j(x_j, \hat{x}, \pi)$$

which corresponds to (2.4). It is worth stressing that condition (2.5) is not needed now; it is implied by the optimality of $\hat{\pi}$ and (2.10). This is an important advantage of the augmented Lagrangian approach over the standard duality, because our ultimate objective is to develop a parallel method in which individual components of the solution vector could be calculated separately. It is, therefore, crucial for us to avoid additional conditions of form (2.5) which inevitably lead to some centralized procedures.

The other advantage is that the dual function (2.7) has much nicer properties. It is concave, piecewise-quadratic and smooth. This allows the application to the dual problem (2.9) of the following steepest ascent method.

**Method of Multipliers**

**Step 1** For fixed multipliers $\pi^k$ find a solution $x^k$ of the problem

$$\min_{l \leq x \leq u} \Lambda(x, \pi^k). \quad (2.12)$$

**Step 2** If $Ax^k = b$ then stop; otherwise set

$$\pi^{k+1} = \pi^k + \gamma \rho (b - Ax^k) \quad (2.13)$$

with some $\gamma \in (0, 2)$, increase $k$ by one and go to Step 1.

The following is the fundamental property of the method of multipliers (see [21, 2]).

**Proposition 3.** If problem (1.1) has an optimal solution then the method of multipliers with $\gamma = 1$ stops after finitely many iterations at an optimal solution $\hat{\pi}$ of the dual problem (2.9).
It follows from Proposition 2 that the primal solution calculated at Step 1 of the method of multipliers is an optimal solution of the primal problem (1.1). For recent references on the application of the method of multipliers to linear programming see [3, 11].

Let us note that we could have easily considered a more general version of the method of multipliers having different penalty coefficients for different rows. This is theoretically equivalent to re-scaling the constraints. Obviously, (2.13) can be then re-scaled accordingly. We shall not consider this version in our theoretical considerations, because it would complicate the notation without any influence on the structure of results, but we shall have scaling in the implementation.

There are two main disadvantages of the method of multipliers: first, Step 1 requires a solution of a quadratic programming problem and, secondly, the augmented Lagrangian does not possess the separability properties of the ordinary Lagrangian. There are many approaches to reformulate some specially structured problems in order to obtain some separability properties of the augmented Lagrangian and decompose Step 1 of the multiplier method. We can mention here the alternating direction methods [8, 9, 10] and the separable approximation developed by Stephanopoulos and Westerberg [26] and used in [27, 4]. The reader is referred to [3] for an extensive discussion of the subject and multiple examples.

In the next section we shall present a Jacobi-type method for the problem of minimizing the augmented Lagrangian (2.6) at Step 1 of the multiplier method. It is close in spirit to the earlier ideas of [26, 27, 4]. Next, adapting the theory developed in [24] we shall show that convergence properties of the method, in particular the speed of convergence, heavily depend on sparsity properties of the matrix $A$ in (1.1).

3. Outline of the method

The main idea of our approach is to carry out under-relaxed nonlinear Jacobi iterations for problem (2.12). If we select the $j$th variable and assume that all other variables $x_s, s \neq j$ are fixed at some values $x^{k,0}_s$, the best value of $x_j$ can be found from the problem

$$\min_{l_j \leq x_j \leq u_j} \Lambda(x_1^{k,0}, \ldots, x_{j-1}^{k,0}, x_j, x_{j+1}^{k,0}, \ldots, x_n^{k,0}, \pi^k).$$

This can be deciphered as

$$\min_{l_j \leq x_j \leq u_j} \tilde{\Lambda}_j(x_j, x^{k,0}, \pi^k)$$

with $\tilde{\Lambda}_j$ defined by (2.11). The regularized Jacobi method can now be stated in detail as follows.

The Regularized Jacobi Method

Initialization. Set $x^{k,0} = x^{k-1}$ and $\nu = 0$.

Dual Phase. Calculate the current primal error $y^{k,0}$ and current prices $p^{k,0}$ by

$$y^{k,0} = b - Ax^{k,0},$$

$$p^{k,0} = \sum_{i=1}^n (c_i + \lambda_i u_i - \lambda_i l_i),$$

$$u_j = \max(l_j, 0),$$

$$l_j = \min(u_j, 0).$$
\[ p^{k,\nu} = \pi^k + \rho y^{k,\nu}. \]  

**Primal Phase.** Calculate the reduced cost

\[ \bar{c}^{k,\nu} = c - A^T p^{k,\nu}. \]  

Solve for \( j = 1, \ldots, n \) subproblems (3.1) getting optimal solutions

\[ \xi_{j}^{k,\nu} = \left[ x_{j}^{k,\nu} - \frac{\bar{c}_{j}^{k,\nu}}{\rho \| A_j \|^2} \right] u_j, \]  

where \([.]^u\) denotes the projection on the interval \([l, u]\). Next, for \( j = 1, \ldots, n \) update the primal variables by

\[ x_{j}^{k,\nu+1} = x_{j}^{k,\nu} + \tau_{j}^{k,\nu} (\xi_{j}^{k,\nu} - x_{j}^{k,\nu}), \]  

where \( \tau_{j}^{k,\nu} \in (0, 1] \), \( j = 1, \ldots, n \) are some under-relaxation parameters.

**Stopping Test.** If \( \xi_{j}^{k,\nu} \neq x_{j}^{k,\nu} \) then increase \( \nu \) by one and go to Dual Phase; otherwise, if \( y_{j}^{k,\nu} \neq 0 \), then set

\[ \pi^{k+1} = \gamma p^{k,\nu} + (1 - \gamma) \pi^k, \]  

\[ x^k = x^{k,\nu}, \text{ increase } k \text{ by one and go to Initialization; otherwise stop.} \]

It is a matter of elementary calculation to show that (3.2)-(3.5) indeed produce a solution to (3.1).

The nonlinear Jacobi method has a long history (see [3] for an extensive discussion of this issue). There are also some works in which it was suggested for minimizing the augmented Lagrangian in a decomposed way. We should mention here the pioneering work of Stephanopoulos and Westerberg [26] (where the results are mostly experimental) and the further works [4, 27, 28].

In general, the Jacobi method has been regarded as slow because available theoretical results (see, e.g., [27]) imply very small values of the under-relaxation parameter (corresponding to \( \tau_j \)'s in (3.6)) to ensure convergence. However, the analysis in [24] shows that for some classes of problems the under-relaxation parameters in the Jacobi method can be relatively large and the speed of convergence quite high. It turns out that performance of the method depends heavily on the sparsity of the constraints. Application to stochastic programming [17] provided a practical evidence for it.

In our case the entire Jacobi method could be boiled down to a series of very simple algebraic operations. This utmost simplicity and good theoretical properties (to be shown in the next sections) provide a solid basis for an efficient implementation, especially in parallel computing environments.

4. **Weak convergence**

Before proceeding to the convergence analysis let us introduce some measures of sparsity of the constraint matrix \( A \). Let \( M_i \) denote the number of nonzeros in the \( i \)th row.
of $A$, $i = 1, \ldots, m$. We define for each column $A_j$ of $A$, $j = 1, \ldots, n$, the average rowcount

$$N_j = \frac{\sum_{i=1}^{m} a_{ij}^2 M_i}{\sum_{i=1}^{m} a_{ij}^2}.$$  \hspace{1cm} (4.1)

In other words, $N_j$ is the weighted average of the number of nonzeros, where the weights are squared column entries. Clearly, there is a simple upper bound on $N_j$:

$$N_j \leq \max\{ M_i : a_{ij} \neq 0 \},$$

which is invariant with respect to scaling of $A$. We assume that all singleton rows (rows with $M_i = 1$) are removed from $A$ in preprocessing, so that $N_j \geq 2$.

We shall show that the $N_j$'s have a direct influence on convergence properties of the method.

Our convergence analysis will use the following algebraic property of sparse matrices.

**Lemma 1.** Let $A$ be an $m \times n$ matrix and let $N_j$, $j = 1, \ldots, n$ be defined by (4.1). Then for every $d \in \mathbb{R}^n$ and every $h \in \mathbb{R}^n$

$$|\langle A d, A h \rangle| \leq \left( \sum_{j=1}^{n} N_j \| A_j \|_2^2 d_j^2 \right)^{\frac{1}{2}} \left( \sum_{j=1}^{n} N_j \| A_j \|_2^2 h_j^2 \right)^{\frac{1}{2}}.$$

**Proof.** Expansion into single entries yields

$$\langle A d, A h \rangle = \sum_{j=1}^{n} \sum_{s=1}^{n} A_j^T A_s d_j h_s = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s \in V(i,j)} (a_{ij} d_j)(a_{is} h_s),$$

where $V(i,j)$ is the set of such $s$ that $a_{ij}a_{is} \neq 0$. Applying Schwarz inequality to the right side of the above equation and noting that $s \in V(i,j)$ if and only if $j \in V(i,s)$ we get

$$\langle A d, A h \rangle^2 \leq \left( \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s \in V(i,j)} a_{ij}^2 d_j^2 \right) \left( \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s \in V(i,j)} a_{ij}^2 h_j^2 \right)$$

$$= \left( \sum_{j=1}^{n} M_i a_{ij}^2 d_j^2 \right) \left( \sum_{j=1}^{n} M_i a_{ij}^2 h_j^2 \right),$$

where in the last relation we used the fact that $V(i,j)$ has $M_i$ elements. The required result follows now from the definition of the $N_j$'s.

Before stating our first convergence theorem, let us note that (3.5) can be equivalently expressed as

$$\zeta_{j}^{k, \nu} = x_{j}^{k, \nu} - \zeta_{j}^{k, \nu} a_{j}^{k, \nu},$$  \hspace{1cm} (4.2)
where \( d_j^{k,\nu} \) is the unconstrained step given by

\[
d_j^{k,\nu} = \frac{c_j^{k,\nu}}{\rho \|A_j\|^2}
\]  

(4.3)

and \( \zeta_j^{k,\nu} \) is the largest coefficient in \([0, 1]\) for which \( \xi_j^{k,\nu} \) remains in \([l_j, u_j]\).

**Theorem 1.** If the under-relaxation coefficients \( \tau_j^{k,\nu} \in (0, 1] \) in (3.6) satisfy the inequalities

\[
0 < \tau_j^{k,\nu} < \frac{2}{\zeta_j^{k,\nu} N_j}, \quad j = 1, \ldots, n,
\]

(4.4)

where \( N_j \) is given by (4.1) (and there is no subsequence of \( \{\tau_j^{k,\nu}\}_{\nu=0}^{\infty} \) convergent to any of the bounds), then

(a) for all \( j = 1, \ldots, n \) we have

\[
\lim_{\nu \to \infty} (\xi_j^{k,\nu} - x_j^{k,\nu}) = 0;
\]

(b) each accumulation point of the sequence \( \{x_j^{k,\nu}\}_{\nu=0}^{\infty} \) is a solution of (2.12).

**Proof.** Let us denote

\[
\tau = \begin{bmatrix}
\tau_1^{k,\nu} \\
\tau_2^{k,\nu} \\
\vdots \\
\tau_n^{k,\nu}
\end{bmatrix}.
\]

In this part of the proof, for clarity, we shall omit the superscripts \( k \) and \( \nu \). By a straightforward calculation, with the use of (3.2)-(3.4),

\[
\Lambda(x + \tau(\xi - x), \pi^k) - \Lambda(x, \pi^k) = e^T \tau(\xi - x) + \frac{1}{2} \rho \|A\tau(\xi - x)\|^2.
\]

Using Lemma 1 with \( d = h = \tau(\xi - x) \) we obtain the inequality

\[
\Lambda(x + \tau(\xi - x), \pi^k) - \Lambda(x, \pi^k) \leq \sum_{j=1}^{n} \tau_j c_j(\xi_j - x_j) + \frac{1}{2} \rho \sum_{j=1}^{n} \tau_j^2 N_j \|A_j\|^2 (\xi_j - x_j)^2.
\]

(4.5)

Using (4.2) and (4.3) in (4.5) we see that for \( \nu = 0, 1, 2, \ldots \) (with the full notation again)

\[
\Lambda(x_j^{k,\nu+1}, \pi^k) - \Lambda(x_j^{k,\nu}, \pi^k) \leq \rho \sum_{j=1}^{n} \tau_j^{k,\nu} (1 - \frac{\tau_j^{k,\nu} \zeta_j^{k,\nu} N_j}{2}) \|A_j\|^2 \zeta_j^{k,\nu} (d_j^{k,\nu})^2.
\]

(4.6)

Thus for \( \tau_j \)'s satisfying (4.4) the sequence \( \{\Lambda(x_j^{k,\nu}, \pi^k)\}_{\nu=0}^{\infty} \) is decreasing and convergent, hence the left side of (4.6) converges to zero. The right side is non-positive, so it converges to zero, too, and assertion (a) is true.
By the definition of $\xi_j^{k,\nu}$,

$$
\xi_j^{k,\nu} = \left[ x_j^k - \frac{\partial \hat{\Lambda}_j(x_j^k, x^k, \pi^k)}{\partial x_j} \right]_i^{u_j},
$$

Let $x^k$ be the limit point of $\{\xi_j^{k,\nu}\}_{\nu=0}^{\infty}$. Passing to the limit in the last equation (over a convergent subsequence) we get

$$
x_j^k = \left[ x_j^k - \frac{\partial \hat{\Lambda}_j(x_j^k, x^k, \pi^k)}{\partial x_j} \right]_i^{u_j}.
$$

Therefore $x^k$ is a solution of (2.12).

Remark. It is clear from our earlier considerations that we can further broaden the bounds (4.4) by defining $N_j$ dynamically in the following way. At first, we determine the set of variables that will not change their values at the current iteration (which have $\xi_j^{k,\nu} = x_j^{k,\nu}$). Then, in calculating $N_j$ by (4.1) we can decrease the counts $M_i$ by ignoring the columns that correspond to the variables which remain unchanged at this iteration.

5. Speed of convergence and strong convergence

We shall now show that the speed of convergence of the Regularized Jacobi Method for solving (2.7) is linear. We shall suppress superscripts $k$ of successive augmented Lagrangians. For the stepsize parameters $\tau_j^{k,\nu}$ and $\xi_j^{k,\nu}$ we suppress the iteration counter $\nu$ as well.

To begin, we develop expressions for the reduced cost and for the optimality gap. Note that the optimal reduced cost $\hat{c}$ is unique over $\hat{x} \in \hat{X}(\pi)$. It is given by

$$
\hat{c} = c - A^T \pi - \rho A^T (b - A \hat{x}) = \hat{\eta} - \hat{\xi},
$$

where

$$
\hat{\eta} \geq 0, \quad \hat{\eta}^T (\hat{x} - l) = 0,
$$

$$
\hat{\xi} \geq 0, \quad \hat{\xi}^T (u - \hat{x}) = 0.
$$

Taking any $\hat{x} \in \hat{X}(\pi)$ and employing (5.1), the reduced cost $\bar{c}$ for $x^\nu$ is

$$
\bar{c} = c - A^T \pi - \rho A^T (b - A x^\nu) = \hat{\eta} - \hat{\xi} + \rho A^T A x^\nu - \hat{\xi}.
$$

For the optimality gap we obtain by direct calculation

$$
\Delta_{\nu} = \Lambda(x^\nu, \pi) - \hat{\Lambda}(\pi)
$$

$$
= (c - A^T (\pi + \rho \hat{b})^T (x^\nu - \hat{x}) + \frac{\rho}{2} \| Ax^\nu \|^2 - \frac{\rho}{2} \| A \hat{x} \|^2
$$

$$
= \frac{\rho}{2} \| A(x^\nu - \hat{x}) \|^2 + \hat{\eta}^T (x^\nu - l) + \hat{\xi}^T (u - x^\nu).
$$
Lemma 2. If the sequence $A_v$, $v = 1, 2, \ldots$, converges to zero, then there exists $\bar{v}$ such that, for all $v \geq \bar{v}$, there is $\hat{x}_v \in \hat{X}(\pi)$ with the property $\hat{x}_v = x_j^*$, for all $j$ such that $x_j^* = l_j$ or $x_j^* = u_j$.

**Proof.** Let $J_L$ and $J_U$ be disjoint subsets of $\{1, \ldots, n\}$. Let us pick any $\hat{x}^* \in \hat{X}(\pi)$. Let $\epsilon(J_L, J_U) = \min_{1 \leq l \leq n} (\rho/2) \| A(x - \hat{x}^*) \|_2^2 + \hat{\eta}^T(x - I) + \hat{\xi}^T(u - x)$ subject to $x_j = l_j, j \in J_L; x_j = u_j, j \in J_U$. If $\epsilon(J_L, J_U) = 0$, there is $\hat{x} \in \hat{X}(\pi)$ with $\hat{x}_j = l_j, j \in J_L; \hat{x}_j = u_j, j \in J_U$. Let $\epsilon$ be the smallest positive $\epsilon(J_L, J_U)$ over all possible sets $J_L$ and $J_U$. By assumption, there exists $\bar{v}$ such that $\Delta_v < \epsilon$, for all $v \geq \bar{v}$, and hence the proof is complete.

**Theorem 2.** Let $N_j$ be defined by (4.1) and $\zeta_j$ by (4.2), for all $j$. Let $\mu \in (0, 2)$ be a constant and define for $j = 1, 2, \ldots$

$$\tau_j = \min \left(1, \frac{\mu}{(\zeta_j N_j)} \right).$$

Then the Regularized Jacobi Method converges linearly:

$$\Delta_{v+1} \leq q \Delta_v$$

with the convergence ratio $q$ such that

$$q \leq \left(1 - \frac{\mu^* \lambda}{\theta} \right)^2 < 1,$$

where $\lambda$ is the smallest nonzero eigenvalue of all matrices $BB^T$, such that $B$ is obtained by taking a nonempty subset of columns of $A$,

$$\theta = \max_{1 \leq j \leq n} \{N_j \| A_j \|_2^2\}$$

and

$$\mu^* = \min(\mu, 1 - \mu/\alpha),$$

with $\alpha \in (\mu, 2)$ chosen in such a way that

$$2 - \alpha = \frac{\lambda \mu^*}{\theta}.$$

**Proof.** Let $J_L$ and $J_U$ be the sets of indices $j$ with $\hat{\eta}_j > 0$ and $\hat{\xi}_j > 0$, respectively. By theorem 1, $\Delta_v \to 0$, so that by (5.3) $A(x^v - \hat{x}) \to 0, \hat{\eta}^T(x^v - I) \to 0$ and $\hat{\xi}^T(u - x^v) \to 0$. Therefore, $\hat{c} \to \hat{\eta} - \hat{\xi}$ in (5.2), $x_j^v \to l_j$, for all $j \in J_L$, $x_j^v \to u_j$, for all $j \in J_U$. Therefore there exists $\bar{v}$ such that, for all $v \geq \bar{v}$ and $j \in J_L \cup J_U$, (employing (3.4), (4.2) and (4.4)), $\tau_j$ is given as

$$\tau_j = \min \left[1, \frac{\mu}{\zeta_j N_j} \right] = \min \left[1, \frac{\mu}{N_j} \max \left(1, \frac{\bar{c}_j}{\rho \| A_j \|_2 (\hat{x}_j - x_j^*)} \right) \right] = 1.$$
Note that \( \tau_j = 1 \) implies that the component \( x_j \) ends up on its bound. Thus, for all \( \nu \geq \bar{\nu} \), \( x^\nu_j = \hat{x}_j \), for all \( j \in J_L \cup J_U \).

From now on we consider the tail of iterations \( \nu \geq \bar{\nu} \), with \( \bar{\nu} \) large enough so that the property of lemma 2 holds as well. In particular, let us study the iteration from \( x^\nu \) to \( x^{\nu+1} \).

We define the sets of "basic" and "nonbasic" variables as follows. We choose some \( \alpha \in (\mu, 2) \) (the exact value of \( \alpha \) will be specified later) and define

\[
J_B = \{ 1 \leq j \leq n : \tau_j < 1 \text{ or } \zeta_j \geq (1 - \mu/\alpha)/N_j \}.
\]

Let us note that in this way we include into \( J_B \) all variables that do not hit a bound at the current iteration. Additionally, we include some variables which hit a bound, provided that the projection in (3.5) did not decrease the steplength too much. Next, \( J_N \) is the set of all other indices \( j \); i.e., for \( j \in J_N \) we have \( \tau_j = 1 \) and \( \zeta_j < (1 - \mu/\alpha)/N_j \).

Observe that \( \tau_j \zeta_j = \mu/N_j \), if \( \tau_j < 1 \), and \( \tau_j \zeta_j \leq \mu/N_j \), otherwise. Therefore, with \( \mu^* \) given by (5.6), we obtain

\[
\frac{\mu^*/N_j}{N_j} \leq \tau_j \zeta_j \leq \mu/N_j, \quad j \in J_B, \tag{5.8}
\]
\[
0 \leq \tau_j \zeta_j \leq (1 - \mu/\alpha)/N_j, \quad j \in J_N. \tag{5.9}
\]

If \( J_B \) is empty, then by lemma 2, the process stops, and there is nothing to prove. Otherwise, denote \( x_B = (x_j)_{j \in J_B} \), \( x_N = (x_j)_{j \in J_N} \), \( B = (A_j)_{j \in J_B} \) and \( N = (A_j)_{j \in J_N} \). Let \( \tau_B \), \( \zeta_B \) and \( S_B \) be diagonal matrices with entries \( \tau_j \), \( \zeta_j \) and \( \|A_j\| \), for \( j \in J_B \), respectively. Similarly for \( J_N \) we define diagonal matrices \( \tau_N \), \( \zeta_N \) and \( S_N \).

By lemma 2 we can choose \( \hat{x} \in \hat{X}(\pi) \) so that \( \hat{x}_j = x_j^{\nu+1} \), for all \( j \) such that \( x_j^{\nu+1} = l_j \) or \( x_j^{\nu+1} = u_j \). By (5.2), (3.5), (4.2) and (4.4), because \( \zeta_j = 0 \) for \( j \in J_L \cup J_U \), we have for all \( j \), by (5.2),

\[
\zeta_j \hat{x}_j = \zeta_j \rho A_j^T A(x^\nu - \hat{x}),
\]

whence

\[
x_j^{\nu+1} - x_j^\nu = \frac{-1}{\rho} S_j^{-2} \tau_j \zeta_j \hat{x}_j \\
= -S_j^{-2} \tau_j \zeta_j A_j^T A(x^\nu - \hat{x}). \tag{5.10}
\]

Since \( x_N^{\nu+1} = \hat{x}_N \), (5.10) implies

\[
B(x_B^{\nu+1} - \hat{x}_B) = B(x_B^\nu - \hat{x}_B) - \bar{B} \bar{B}^T (B, N)(x^\nu - \hat{x}). \tag{5.11}
\]

and

\[
0 = N(x_N^\nu - \hat{x}_N) - \bar{N} \bar{N}^T (B, N)(x^\nu - \hat{x}), \tag{5.12}
\]

where

\[
\bar{B} = BS_B^{-1/2} \zeta_B^{1/2}, \\
\bar{N} = NS_N^{-1/2} \zeta_N^{1/2}.
\]

Let us define the matrix

\[
E = (I - \bar{N} \bar{N}^T).
\]
From (5.12) we obtain

$$E(B, N)(x^\nu - \hat{x}) = B(x_B^\nu - \hat{x}_B).$$  \hspace{1cm} (5.13)

Let us now make an important observation that explains our definitions of $J_B$ and $J_N$. By lemma 1 and (5.9) for every $u$ we have

$$u^T \tilde{N}^T \tilde{N} u \leq \sum_{j \in J_N} N_j \|A_j\|^2 \frac{\tau_j \xi_j}{\|A_j\|^2} u_j^2 \leq (1 - \mu/\alpha)\|u\|^2.$$  \hspace{1cm} (5.14)

Therefore all eigenvalues of $\tilde{N}^T \tilde{N}$ are in $[0, 1 - \mu/\alpha]$. The nonzero eigenvalues of $\tilde{N}^T \tilde{N}$ and $\tilde{N} \tilde{N}^T$ are the same. Hence, the eigenvalues of $E$ are in $[\mu/\alpha, 1)$. Consequently, $E$ is nonsingular. Denote the maximum eigenvalue of $E$ by $\lambda_E$.

The optimality gap in (5.3) simplifies into

$$\Delta_\nu = \frac{\rho}{2} \|e^\nu\|^2,$$  \hspace{1cm} (5.15)

where $e^\nu = A(x^\nu - \hat{x})$ is the primal feasibility error. By (5.13) we have

$$e^\nu = E^{-1}e^\nu = E^{-1}Bu,$$  \hspace{1cm} (5.16)

with

$$u = S_B \tau_B^{-1/2} \xi_B^{-1/2} (x_B^\nu - \hat{x}_B).$$

Summing equations (5.11) and (5.12) and using (5.16) we get

$$e^{\nu+1} = (E - \bar{B}\bar{B}^T)E^{-1}Bu.$$  \hspace{1cm} (5.17)

Therefore, by (5.15), (5.16) and (5.17), with $Q = E^{-1/2}\bar{B}$, we obtain

$$\frac{\Delta_{\nu+1}}{\Delta_\nu} = \frac{\|E^{1/2}(I - QQ^T)Qu\|^2}{\|E^{-1/2}Qu\|^2} \leq \bar{\lambda}_E^2 \frac{\|I - QQ^T\|}{\|Qu\|^2} \leq \bar{\lambda}_E^2 \max(1 - \lambda_t)^2,$$  \hspace{1cm} (5.18)

where parameters $\lambda_t$ refer to all nonzero eigenvalues of $QQ^T$. Let $\bar{\lambda}_B$ denote the largest eigenvalue of $\bar{B}\bar{B}^T$, and recall that the smallest eigenvalue of $E$ is at least $\mu/\alpha$. Then $\lambda_t \leq \alpha \bar{\lambda}_B/\mu$, for all $t$. Similarly to (5.14), lemma 1 with (5.8) yield $\bar{\lambda}_B \leq \mu$. Therefore, $\lambda_t \leq \alpha$. Hence, by (5.18) and because $\lambda_E \leq 1$

$$\frac{\Delta_{\nu+1}}{\Delta_\nu} \leq \max \left[ (1 - \alpha)^2, \bar{\lambda}_E^2 (1 - \lambda_{Q}^{\min})^2 \right].$$  \hspace{1cm} (5.19)

with $\lambda_{Q}^{\min} = \min_t \lambda_t$.

Let $\lambda_{B}^{\min}$ and $\lambda_{B}^{\max}$ denote the smallest positive eigenvalue of $\bar{B}\bar{B}^T$ and $BB^T$, respectively. Then $\lambda_{Q}^{\min} \geq \lambda_{B}^{\min}/\lambda_E$. Consequently,

$$\bar{\lambda}_E^2 (1 - \lambda_{Q}^{\min})^2 \leq (\bar{\lambda}_E - \lambda_{B}^{\min})^2.$$  \hspace{1cm} (5.20)
With \( \mu^* \) and \( \theta \) given by (5.6), employing (5.8) we have
\[
\mu \geq \bar{\lambda}_B \geq \lambda_B^{\min} \geq \lambda_B^{\min \ j}(S_j^2 r_j q_j) \geq \frac{\mu^* \lambda_B^{\min}}{\theta} \geq \frac{\mu^* \lambda}{\theta} > 0. \tag{5.21}
\]
In summary, \( \bar{\lambda}_E \in [\mu, 1] \) and \( \lambda_B^{\min} \in [\mu^* \lambda/\theta, \mu] \). Therefore by (5.20),
\[
\bar{\lambda}_E^2 \left(1 - \lambda_Q^{\min}\right)^2 \leq (1 - \frac{\mu^* \lambda}{\theta})^2 < 1. \tag{5.22}
\]
By (5.19) and (5.22)
\[
q \leq \max \left((1 - \alpha)^2, (1 - \frac{\mu^* \lambda}{\theta})^2 \right). \tag{5.23}
\]
We shall now show that a solution \( \alpha^* \) to (5.7) exists such that \( \alpha^* \in (\mu, 2) \). Indeed, at \( \alpha = \mu \) the left hand side of (5.7) is positive and the right hand side equals 0, while at \( \alpha = 2 \) we have the converse relation. For \( \alpha = \alpha^* \) both cases at the right side of (5.23) are equal and we get the desired expression for the ratio \( q \). The proof is complete.

Let us note that if the columns of \( A \) are normalized, the parameter \( \theta \) in (5.5) equals just \( \max_j N_j \), while \( \lambda \) can be interpreted as the inverse of a generalized condition index of \( A \).

We are now ready to prove strong convergence of the Regularized Jacobi Method.

**Theorem 3.** Adopt definitions of theorem 2. Then the sequence \( \{x^\nu\} \) is convergent to a solution \( \hat{x} \in \hat{X}(\pi) \) and there is a constant \( C \) such that
\[
\|x^\nu - \hat{x}\| \leq C q^{\nu/2}, \, \nu = 1, 2, \ldots,
\]
where \( q \in (0, 1) \) is defined by (5.5).

**Proof.** Let \( \nu \) be defined as in the proof of theorem 2. For \( \nu \geq \nu \) from (5.10) and (5.15) we obtain
\[
|x_j^{\nu+1} - x_j^\nu| \leq \frac{\|A(x^\nu - \hat{x})\|}{\|A_j\|} = \frac{(2\Delta \nu / \rho)^{1/2}}{\|A_j\|}.
\]
Then by (5.4)
\[
\|x^{\nu+1} - x^\nu\| \leq C_1 q^{\nu/2}
\]
with some constant \( C_1 \). It follows that \( \{x^\nu\} \) is a Cauchy sequence and the distance to its limit \( \hat{x} \) can be estimated as
\[
\|x^\nu - \hat{x}\| \leq C_1 \sum_{\nu=1}^{\infty} q^{1/2} = C_1 q^{1/2} \frac{q^{\nu/2}}{1 - q^{1/2}}.
\]
By theorem 1, the limit is in \( \hat{X}(\pi) \). The proof is complete.

Strong convergence of nonlinear Jacobi iterations has been earlier proved in [20], but with small stepsizes (of order \( 1/n \)). Our analysis provides more practical stepsize bounds and rate of convergence estimates relating them to the numbers of nonzeros in rows, which are for large problems orders of magnitude smaller than \( n \).
6. Implementation

An experimental computer code has been developed on the basis of the theoretical results presented in this paper and an earlier Gauss-Seidel code of [14]. The initial experiments have immediately shown that scaling is crucial to make the method work in practice. It has also proved inefficient to carry out primal iterations until the optimality of each one of the augmented Lagrangian problems is obtained. In fact, our experiments indicate (see the following section), that just one primal iteration after each dual update is justified, given that steps for updating the dual multipliers are suitably under-relaxed.

We consider alternative definitions for stepsize in updating the primal variables. In particular, a constant stepsize is compared against a safe stepsize $\tau_j$ defined by (4.4). We also experiment with versions, where the primal steps defined by the former two variants are used only for determining a search direction, and updating primal variables is based on line search optimization in that direction. For line search, two alternatives are tested: first, unconstrained line search followed by under-relaxation and projection (to meet the bounds on primal variables), and constrained line-search optimization. Experience with all such variants will be discussed in the following section.

We shall call our implementation Jacobi. An iteration in Jacobi consists of a dual phase, a primal phase and, depending on the variant, possibly of a line search (along the direction specified in the primal phase). It should be stressed that all phases of the method have a great potential for parallelization. The dual phase can be carried out simultaneously for all rows. Revised prices are then submitted to processors of the primal phase. After the dual phase is completed for all rows, all primal variables can be processed in parallel. The line search can be done in parallel as well. The line search stepsize can be shared by primal processors to update primal variables, and these values can be subsequently sent to dual processors. The dual phase starts after all primal variables have been updated. Let us note that sparsity can be exploited in communication as well. The iterations end when optimality conditions are met within specified tolerances, or if an iterations limit is met.

It was pointed out that scaling is crucial for an efficient performance of the algorithm. However, any scaling on the top of the one employed in our implementation is unnecessary, because (excluding the minimum tolerance parameter to be introduced shortly), our approach is independent of scaling. This means that unless the minimum tolerance becomes active, the iterations in our algorithm produce the same sequence of solutions for all problems that are obtained by scaling primal and dual variables.

We shall discuss each one of the three components of Jacobi in more detail shortly. However, we begin with feasibility tolerances, penalty parameters and data structure employed in Jacobi.

Feasibility tolerances

A positive dual tolerance vector $\delta = (\delta_j) \in \mathbb{R}^n$ depending on the current solution will be applied while checking dual feasibility and complementarity: a reduced cost $\tilde{c_j}$ vanishes, if it is equal to zero within tolerance $\delta_j$. Similarly, a positive primal tolerance vector $\epsilon = (\epsilon_i) \in \mathbb{R}^m$ depending on the current solution is employed to test
primal feasibility. Tolerances $\delta$ and $\epsilon$ are defined for each iteration as follows. First, a reference primal vector $Q = (Q_i) \in \mathbb{R}^m$ and a reference dual vector $V = (V_j) \in \mathbb{R}^n$ are defined so that

$$Q_i = |b_i| + \sum_{j=1}^{n} |x_{ij}a_{ij}|,$$

$$V_j = |c_j| + \sum_{i=1}^{m} |p_ia_{ij}|,$$

where $x_{ij}$ and $p_i$ denote primal and dual solutions at the beginning of the current iteration. We then define auxiliary tolerance vectors $\delta'$ and $\epsilon'$ equal to a fraction of reference vectors $V$ and $Q$, respectively:

$$\delta' = \phi V \quad \text{and} \quad \epsilon' = \phi Q,$$

where $\phi$ is a relative tolerance. Let $\sigma$ be a minimum tolerance (both for primal and dual) and denote by $\delta^-$ and $\epsilon^-$ the tolerances at the beginning of the iteration. For the first iteration, such initial values of all primal and dual tolerances are set to $\sigma$. Finally, the updated tolerances $\delta$ and $\epsilon$ for the current iteration are computed as

$$\delta_j = \max(\sigma, \alpha\delta'_j + (1 - \alpha)\delta^-_j),$$

$$\epsilon_j = \max(\sigma, \alpha\epsilon'_j + (1 - \alpha)\epsilon^-_j),$$

where $\alpha$ is an exponential smoothing parameter. Its purpose is to prevent erratic behavior of tolerances over the iterations.

**Penalty Parameters and Scaling**

We shall denote by $D_i$ the penalty parameter associated with row $i$. Let $D$ be the diagonal matrix with diagonal elements $D_i$. Given a primal tolerance vector $\epsilon > 0$, one might attempt to solve the LP problem by introducing large values for $D_i$ and solving the augmented Lagrangian problem for some $\pi$. If the resulting optimum price vector is $p$, then the primal error $y = D^{-1}(p - \pi)$ tends to meet the tolerance $\epsilon$. However, large penalty parameters imply very small primal steps. Therefore, this approach is extremely inefficient.

In order to cure this handicap, several approaches may be considered. First, following [7], one might solve a sequence of equilibrium problems gradually enlarging the penalty parameters until primal tolerance is met. This approach suffers from serious inefficiency as well. Second, following [12] and [22], we may solve a sequence of augmented Lagrangian problems with maintaining the penalty parameters unchanged, but shifting the dual multipliers $\pi_i$ from one problem to another: if at the optimum of an augmented Lagrangian primal feasibility is violated, for some $i$, then for the subsequent problem we increase $\pi_i$ by $\gamma_i y_i D_i$ (cf. (2.13)), where $\gamma_i$ is an under-relaxation parameter and $y_i$ is the primal infeasibility. Based on our experience, this approach performs well provided that one learns at first how to choose the penalty parameters. Again, there is a risk that if parameters $D_i$ are too large we end up with small inefficient steps. On the
other hand, if these elements are too small and optimization of augmented Lagrangians is not carried to the end, then the sequence of solutions generated in the course of the iterations does not necessarily converge.

We shall apply our earlier approach of [14] and factorize $D_i$ into $\rho S_i$. Here $\sqrt{S_i}$ serves as a scaling factor for row $i$ and the penalty factor $\rho$ is the penalty parameter in (2.6) (applied to the problem with rows scaled by $\sqrt{S_i}$). Again, let $\delta_j$ and $\epsilon_i$ be the dual and primal tolerances. The penalty parameter is then defined following [14]:

$$D_i = \rho E_j \left\{ \frac{\delta_j}{|a_{ij}|\epsilon_i} \right\},$$

(6.1)

where operator $E_j$ refers to an average over $j$, for $a_{ij}$ nonzero. Initially we experimented with $E_j$ being an arithmetic mean. However, a geometric mean proved to result in a more robust implementation. Finally we ended up with a harmonic mean, which is cheaper to compute and provides a similar robustness as the geometric mean.

Along with the tolerances, also the penalty parameters will be updated in the course of the iterations. We shall first apply (6.1) for obtaining auxiliary penalty parameters. The penalty parameters employed in Jacobi are then obtained via exponential smoothing over iterations. Again, we employ the weight $\alpha$ for the auxiliary parameter and $1 - \alpha$ for the parameter employed in the preceding iteration. For the first iteration such initial values are set equal to zero.

**The Data Structure**

The data in $A$ and $c$ are stored columnwise accounting for sparsity. For the purpose of dual updates, the locations of nonzero elements of $A$ and $c$ are stored row-wise as well. Bounds $l$ and $u$ are stored as dense vectors. Vector $b$ is stored in bounds for the logical variables.

**Dual Phase**

To begin the first iteration, we set all dual multipliers $\pi_i$ to zero. The dual update of Jacobi consists of the following steps for each row $i$.

1. Update the penalty parameters, primal tolerances and dual tolerances of the logical (slack) variables, and compute the primal infeasibility (3.2).
2. Optimality test is carried out for primal feasibility, and for dual feasibility and complementarity of the slacks for inequalities.
3. Update dual multipliers $\pi$ according to (3.7) with under-relaxation. The under-relaxation parameter $\gamma$ is constant over the iterations, unless the relative primal error $\mu_i = |y_i|/\epsilon_i < \beta$ where $\beta$ is a dual update threshold. In the latter case, $\gamma$ is scaled down by the factor $\mu_i/\beta$ (the significance of this detail is tested in section 7.3). For inequality rows, $\pi_i$ is projected for correct sign.
4. The price is updated by (3.3), and for inequalities, projected as well.
Primal Phase

To begin the first iteration we set all primal variables equal to zero and project onto the bounds. The primal update, for each activity $j$, is as follows.

1. Evaluate the reduced cost $\bar{c}_j$ by (3.4), update the dual tolerances and compute $A_j^TDA_j$.

2. Perform the optimality test, i.e. dual feasibility and complementarity test, for column $j$.

3. Determine the direction $h_j$: Let $x_j$ be the level of activity $j$ at the beginning of the iteration and $\tau_j$ a stepsize parameter. We define

   $$h_j = \tau_j \zeta_j d_j,$$

   where $\zeta_j$ and $d_j$ are given by (4.2) and (4.3). However, if the relative dual error $|\bar{c}_j|/\delta_j < \eta$, we set $h_j = 0$. Here $\eta$ is a primal update threshold.

4. Determine the updated primal solution: if no line search is considered, then the updated activity level is $x_j + h_j$. Otherwise, a line search is carried out employing the direction $h = (h_j)$, and the primal variables obtain their revised values thereafter.

The stepsize $\tau_j$ is either constant for all $j$, or it is given by $\tau_j = \min(1, 1/\zeta_j N_j)$ following (4.4). Such safe stepsize is computed taking into account only active rows and columns while determining $N_j$. A column is considered active at a certain iteration, if it has failed the optimality test (at least once) during a lag period of iterations. Similarly a row is considered active, if it is of equality type or if it has been binding (at least once) over the same lag period. Also in the dual phase, only active columns are taken into account for computing auxiliary penalty parameters.

Line Search (Optional)

If a line search is applied in the direction $h$, an updated primal vector is obtained by taking an under-relaxed step toward the unconstrained line search optimum and projecting onto the simple bounds. A line search under-relaxation parameter is denoted by $\omega$. Alternatively, we may perform a constrained line search, whereby we check the feasibility with respect to the bounds on primal variables while moving in the direction $h$. In this case no under-relaxation is applied. The computations for the line search are distributed to primal and dual processors.

7. Computational Tests

Jacobi and some of its variants were tested on 68 problems from the Netlib library. The set is the same as the one used in our earlier study [14] concerning Gauss-Seidel type
of iterations for serial computing. Tables 1 and 2 show the names and dimensions of these problems. The serial run times (excluding input and output) in a HP9000/720 for Minos 5.1 using default values for specs parameters are reported as reference times to be used for efficiency comparisons. For Jacobi, simulation runs were performed on the same computer to obtain estimates of parallel run times.

The outline of this section is as follows: we discuss first the base case of Jacobi and compare the results with Minos 5.1. Secondly, the performance of the safe stepsize of Theorem 1 is compared with a constant stepsize employed in the base case. Thereafter, constrained line search is applied to the two stepsize variants of Jacobi. Subsequently, these variants without line search are tested as well. Sensitivity of the base case of Jacobi with respect to various program parameters is tested by altering a single parameter at a time. The impact of the number of iterations between updates of multipliers $\pi$ is studied thereafter. Finally, a comparison of Jacobi against a Gauss-Seidel approach [14] and against an interior point algorithm [1] is reported.

### 7.1 The Jacobi Base Case

For the relative tolerance $\phi$, two values are applied in the tests: $\phi = 0.01$ and $\phi = 0.001$. A minimum tolerance $\sigma = 0.1\phi$, both for primal and dual constraints, is applied. Initial values for primal variables $x_j$ and for dual multipliers $\pi$, are all equal to zero; initial penalty parameters $D_i$ are all equal to zero. For determining active rows and columns, the iteration count for the lag is $\theta m$, with $\theta = 10$. A maximum iteration limit is employed with values 50,000 and 300,000, for $\phi = 0.01$ and $\phi = 0.001$, respectively. Other default parameters of Jacobi are given in Table 3.

Denote by $t_s$ the serial run time obtained by a simulation run. Let $\lambda$ be the share of $t_s$ taken by the primal phase (including line search computations allocated to primal processors) and $1 - \lambda$ the corresponding share of the dual phase. The share $\lambda$ is measured for each test problem separately, ranging from 24% to 56%. Thus for parallel computations, omitting communication time among processors and assuming that there is at least one processor for each row and column, $\lambda t_s/n$ is a measure of the parallel run time for primal updates and $(1 - \lambda) t_s/m$ is that for the dual. Thereby we obtain a measure of the parallel run time $t_p$:

$$t_p = t_s \left( \frac{\lambda}{n} + \frac{1 - \lambda}{m} \right). \quad (7.1)$$

This formula assumes that all processors (in a particular phase, primal or dual) are loaded with equal tasks, so that the execution time is the same for all. Alternatively, we might define the parallel run time based on the worst cases (the longest run times of primal and dual phases). Our measurement indicates that this definition yields a run time which is 3.5 times the one given by (7.1), on the average. In the sequel we shall employ (7.1) for the following reasons. The worst case results from the increase of the cost in the dual phase due to unequal distribution of nonzeros over the rows. Nineteen of the test problems contain rows (excluding the objective row) with more than 100 nonzeros (the maximum being 1477 for woodw). The main effort for the dual phase is to compute three vector inner products (with dimension equal to the nonzero
<table>
<thead>
<tr>
<th>Problem</th>
<th>Rows</th>
<th>Columns</th>
<th>Nonzeros</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>80bau3b</td>
<td>2263</td>
<td>9799</td>
<td>29063</td>
<td>272.9</td>
</tr>
<tr>
<td>stocfor2</td>
<td>2158</td>
<td>2031</td>
<td>9492</td>
<td>37.1</td>
</tr>
<tr>
<td>degen3</td>
<td>1504</td>
<td>1818</td>
<td>26230</td>
<td>147.9</td>
</tr>
<tr>
<td>sctap3</td>
<td>1481</td>
<td>2480</td>
<td>10734</td>
<td>11.6</td>
</tr>
<tr>
<td>pilot</td>
<td>1442</td>
<td>3352</td>
<td>43220</td>
<td>1153.0</td>
</tr>
<tr>
<td>ganges</td>
<td>1310</td>
<td>1681</td>
<td>7021</td>
<td>7.5</td>
</tr>
<tr>
<td>sierra</td>
<td>1228</td>
<td>2036</td>
<td>9252</td>
<td>13.6</td>
</tr>
<tr>
<td>ship12l</td>
<td>1152</td>
<td>5427</td>
<td>21597</td>
<td>12.7</td>
</tr>
<tr>
<td>ship12s</td>
<td>1152</td>
<td>2763</td>
<td>10941</td>
<td>5.2</td>
</tr>
<tr>
<td>woodw</td>
<td>1099</td>
<td>8405</td>
<td>37478</td>
<td>89.0</td>
</tr>
<tr>
<td>sctap2</td>
<td>1091</td>
<td>1880</td>
<td>8124</td>
<td>7.2</td>
</tr>
<tr>
<td>scfxm3</td>
<td>991</td>
<td>1371</td>
<td>7846</td>
<td>12.9</td>
</tr>
<tr>
<td>pilotnov</td>
<td>916</td>
<td>2172</td>
<td>13129</td>
<td>32.9</td>
</tr>
<tr>
<td>pilot_ja</td>
<td>941</td>
<td>1988</td>
<td>14706</td>
<td>105.5</td>
</tr>
<tr>
<td>czprob</td>
<td>930</td>
<td>3523</td>
<td>14173</td>
<td>16.9</td>
</tr>
<tr>
<td>25fv47</td>
<td>822</td>
<td>1571</td>
<td>11127</td>
<td>89.6</td>
</tr>
<tr>
<td>ship08l</td>
<td>779</td>
<td>4283</td>
<td>17085</td>
<td>4.7</td>
</tr>
<tr>
<td>ship08s</td>
<td>779</td>
<td>2387</td>
<td>9501</td>
<td>2.1</td>
</tr>
<tr>
<td>pilot_we</td>
<td>723</td>
<td>2789</td>
<td>9218</td>
<td>56.2</td>
</tr>
<tr>
<td>nesm</td>
<td>663</td>
<td>2923</td>
<td>13988</td>
<td>27.6</td>
</tr>
<tr>
<td>scfxm2</td>
<td>661</td>
<td>914</td>
<td>5229</td>
<td>6.3</td>
</tr>
<tr>
<td>perold</td>
<td>626</td>
<td>1376</td>
<td>6026</td>
<td>32.3</td>
</tr>
<tr>
<td>gfrd_pnc</td>
<td>617</td>
<td>1092</td>
<td>3467</td>
<td>3.6</td>
</tr>
<tr>
<td>shell</td>
<td>537</td>
<td>1775</td>
<td>4900</td>
<td>1.66</td>
</tr>
<tr>
<td>fffff800</td>
<td>525</td>
<td>854</td>
<td>6235</td>
<td>6.08</td>
</tr>
<tr>
<td>agg2</td>
<td>517</td>
<td>302</td>
<td>4515</td>
<td>1.16</td>
</tr>
<tr>
<td>agg3</td>
<td>517</td>
<td>302</td>
<td>4531</td>
<td>1.22</td>
</tr>
<tr>
<td>seba</td>
<td>516</td>
<td>1028</td>
<td>4874</td>
<td>1.96</td>
</tr>
<tr>
<td>scrs8</td>
<td>491</td>
<td>1169</td>
<td>4029</td>
<td>3.08</td>
</tr>
<tr>
<td>agg</td>
<td>489</td>
<td>163</td>
<td>2541</td>
<td>.84</td>
</tr>
<tr>
<td>scagr25</td>
<td>472</td>
<td>500</td>
<td>2029</td>
<td>1.86</td>
</tr>
<tr>
<td>standmps</td>
<td>468</td>
<td>1075</td>
<td>3686</td>
<td>1.20</td>
</tr>
<tr>
<td>grow22</td>
<td>441</td>
<td>946</td>
<td>8318</td>
<td>7.80</td>
</tr>
<tr>
<td>pilot4</td>
<td>411</td>
<td>1000</td>
<td>5145</td>
<td>11.08</td>
</tr>
<tr>
<td>ship04l</td>
<td>403</td>
<td>2118</td>
<td>8450</td>
<td>1.32</td>
</tr>
<tr>
<td>ship04s</td>
<td>403</td>
<td>1458</td>
<td>5810</td>
<td>.66</td>
</tr>
<tr>
<td>etamacro</td>
<td>401</td>
<td>688</td>
<td>2489</td>
<td>1.70</td>
</tr>
</tbody>
</table>

Table 1: Number of rows, columns and non-zeros for Netlib test problems. Time is the reference time (sec) obtained by Minos 5.1.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Rows</th>
<th>Columns</th>
<th>Nonzeros</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>scsd8</td>
<td>398</td>
<td>2750</td>
<td>11334</td>
<td>25.86</td>
</tr>
<tr>
<td>scorpion</td>
<td>389</td>
<td>358</td>
<td>1708</td>
<td>.36</td>
</tr>
<tr>
<td>standgub</td>
<td>362</td>
<td>1184</td>
<td>3147</td>
<td>.36</td>
</tr>
<tr>
<td>standata</td>
<td>360</td>
<td>1075</td>
<td>3038</td>
<td>.36</td>
</tr>
<tr>
<td>stair</td>
<td>357</td>
<td>467</td>
<td>3857</td>
<td>3.66</td>
</tr>
<tr>
<td>tuff</td>
<td>334</td>
<td>587</td>
<td>4523</td>
<td>4.44</td>
</tr>
<tr>
<td>scfxml</td>
<td>331</td>
<td>457</td>
<td>2612</td>
<td>1.32</td>
</tr>
<tr>
<td>bandm</td>
<td>306</td>
<td>472</td>
<td>2659</td>
<td>1.92</td>
</tr>
<tr>
<td>sctapl</td>
<td>301</td>
<td>480</td>
<td>2052</td>
<td>.62</td>
</tr>
<tr>
<td>grow15</td>
<td>301</td>
<td>645</td>
<td>5665</td>
<td>3.74</td>
</tr>
<tr>
<td>capri</td>
<td>272</td>
<td>353</td>
<td>1786</td>
<td>.58</td>
</tr>
<tr>
<td>bore3d</td>
<td>234</td>
<td>315</td>
<td>1525</td>
<td>.46</td>
</tr>
<tr>
<td>e226</td>
<td>224</td>
<td>282</td>
<td>2767</td>
<td>1.26</td>
</tr>
<tr>
<td>brandy</td>
<td>221</td>
<td>249</td>
<td>2150</td>
<td>1.38</td>
</tr>
<tr>
<td>sc205</td>
<td>206</td>
<td>203</td>
<td>552</td>
<td>.26</td>
</tr>
<tr>
<td>vtp.base</td>
<td>199</td>
<td>203</td>
<td>914</td>
<td>.12</td>
</tr>
<tr>
<td>israel</td>
<td>175</td>
<td>142</td>
<td>2358</td>
<td>.70</td>
</tr>
<tr>
<td>beaconfd</td>
<td>174</td>
<td>262</td>
<td>3476</td>
<td>.26</td>
</tr>
<tr>
<td>forplan</td>
<td>162</td>
<td>421</td>
<td>4916</td>
<td>.96</td>
</tr>
<tr>
<td>scsd6</td>
<td>148</td>
<td>1350</td>
<td>5666</td>
<td>3.24</td>
</tr>
<tr>
<td>grow7</td>
<td>141</td>
<td>301</td>
<td>2633</td>
<td>.76</td>
</tr>
<tr>
<td>scagr7</td>
<td>130</td>
<td>140</td>
<td>553</td>
<td>.14</td>
</tr>
<tr>
<td>stocforl</td>
<td>118</td>
<td>111</td>
<td>474</td>
<td>.10</td>
</tr>
<tr>
<td>share1b</td>
<td>118</td>
<td>225</td>
<td>1182</td>
<td>.50</td>
</tr>
<tr>
<td>sc105</td>
<td>106</td>
<td>103</td>
<td>281</td>
<td>.10</td>
</tr>
<tr>
<td>share2b</td>
<td>97</td>
<td>79</td>
<td>730</td>
<td>.12</td>
</tr>
<tr>
<td>recipe</td>
<td>92</td>
<td>180</td>
<td>752</td>
<td>.04</td>
</tr>
<tr>
<td>scsd1</td>
<td>78</td>
<td>760</td>
<td>3148</td>
<td>.68</td>
</tr>
<tr>
<td>adlittle</td>
<td>57</td>
<td>97</td>
<td>465</td>
<td>.06</td>
</tr>
<tr>
<td>sc50a</td>
<td>51</td>
<td>48</td>
<td>131</td>
<td>.02</td>
</tr>
<tr>
<td>afiro</td>
<td>28</td>
<td>32</td>
<td>88</td>
<td>.004</td>
</tr>
</tbody>
</table>

Table 2: Number of rows, columns and non-zeros for Netlib test problems. Time is the reference time (sec) obtained by Minos 5.1.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>0.5</td>
<td>Penalty factor</td>
</tr>
<tr>
<td>$\omega$</td>
<td>0.5</td>
<td>Line search under-relaxation</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.5</td>
<td>Exponential smoothing weight</td>
</tr>
<tr>
<td>$\tau$</td>
<td>0.1</td>
<td>Primal stepsize</td>
</tr>
<tr>
<td>$\eta$</td>
<td>1.0</td>
<td>Primal update threshold</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.1</td>
<td>Dual stepsize</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1.0</td>
<td>Dual update threshold</td>
</tr>
</tbody>
</table>

Table 3: Default values of parameters

Such tasks could be further parallelized, for instance, by sharing the capacity of processors which are less heavily loaded. Therefore, we believe that formula (7.1) serves as a better indicator of the potential of Jacobi.

We define the speedup of Jacobi against another method as the serial run time of the latter divided by the parallel run time $t_p$ of Jacobi. The simplex method and interior point methods do not suit for parallel computation the way Jacobi does. Therefore for these algorithms, we use the serial run time in the comparisons. Obviously, there is some gain from parallel computation for the non-Jacobi methods as well. Besides, communication time is omitted for Jacobi. Therefore our speed-ups should be regarded as optimistic.

Speed-ups for the two tolerances $\phi = 0.01$ and $\phi = 0.001$ of Jacobi base case against Minos 5.1 are depicted in Figure 1 as a function of problem complexity which is measured by the time required by Minos. In Figure 2 the same speed-ups are shown as a function of (the inverse of) problem density (the share of nonzeros in matrix $A$), a measure which is readily available from problem data. The general tendency in these results seems to be that the larger the problem in terms of complexity the larger is the speed-up in favor of Jacobi. For relative tolerance $\phi = 0.01$ and $\phi = 0.001$, a linear regression on logarithms, which is depicted in Figure 1 as well, indicates that the speed-up increases by a factor of 1.6 when the Minos time doubles. A similar regression in Figure 2 implies an increase in speed-up by a factor 2.0 when the density decreases by fifty percent.

Serial run times, iteration counts and relative errors in the objective function value for Jacobi base case are reported in Tables 4 and 5 and depicted in Figures 3-5, respectively. Omitted figures in Tables 4 and 5 refer to cases which did not converge within the iteration limit. A general observation in Figure 3 is that the parallel run time as a function of problem complexity increases slowly: when the Minos time doubles the run time for Jacobi increases by 27 percent only. Similarly, doubling of Minos time implies a 20 percent increase in the number of iterations (see Figure 4).

For the entire set of 68 test problems, the average run time for Jacobi was 0.18 sec.
and 0.85 sec., and the average number of iterations was 4900 and 23000, for relative tolerances $\phi = 0.01$ and $\phi = 0.001$, respectively. Thus the precision requirement has a significant impact on computational effort: decreasing relative tolerances by a factor of ten increases the solution time approximately by a factor of five.

The error in the objective function is typically of the same order of magnitude as the relative tolerance $\phi$. Thinking, for instance, of economic applications of linear programming, the precision obtained by Jacobi appears quite satisfactory in most cases, in particular, for $\phi = 0.001$. There are some notable exceptions, like problems tuff and forplan, however. In most nonconvergent cases the accuracy obtained at the iteration limit appears quite satisfactory; see problems woodw and etamacro, for example. In such cases it turns out that the algorithm reaches the neighborhood of the optimum rather fast, but then it fails to meet dual and/or primal feasibility for a few variables and/or constraints. Avoiding nonconvergence and poor precision in such cases requires further investigation.

Figures 6-10 illustrate the behavior of Jacobi over the iterations. The largest test problem in Table 1, called 80bau3b, was chosen as an example, with relative tolerance $\phi = 0.001$. Figure 6 shows the relative error of the objective function value over the iterations. At the end, after 28591 iterations, an error of 0.06% is obtained. The same level is reached already in about 2000 iterations. The long tail, characteristic to Jacobi, comprising of 90% of iterations is needed to meet the optimality conditions. Figures 7 and 8 show the maximal primal and dual errors (relative to tolerances) and the number of infeasibilities. In the tail, the relative errors decrease slowly while the number of infeasibilities is small as compared to the numbers of variables (9799) and constraints (2263). This too is typical for Jacobi. The run was made with a constant stepsize. However for the illustration also the safe stepsize $\tau_j$ was computed at each iteration. Figure 9 shows the minimum, maximum and average value of the safe stepsize over the iterations, and Figure 10 indicates the number of active rows and columns (in percent relative to the number of constraints). The results confirm our theoretical expectations; the safe stepsizes are quite large and they quickly grow in the final stage of the algorithm when the numbers of active rows and columns decrease.

The largest Netlib problem, called stocfor3, has 16676 rows, 15695 columns, and 74004 non-zeros. It was successfully solved with Jacobi: for relative tolerance equal to 0.01, the run time $t_p$ was 0.092 sec and the relative error in the objective function value was 0.044%. Two additional energy-economy models, called o731u and o748, were obtained from IIASA. Their respective dimensions $m \times n$ are 6479 $\times$ 4585 and 5171 $\times$ 4015, and the numbers of nonzero elements in $A$ are 37269 and 23862. For relative tolerance $\phi = 0.01$, the run time $t_p$ was 0.220 sec and 0.132 sec for o731u and o748, respectively, and the relative errors in the objective function 0.007 % and 0.056 %.

### 7.2 Stepsize and Line Search Variants

In Theorem 1 the safe stepsize for primal updates was introduced. We shall now compare the impact of this rule relative to the base case, where a constant step size is applied to all primal variables. With respect to line search, we shall compare three
Table 4: Parallel run time (msec), number of iterations and relative error in the objective function value for Jacobi base case; $\phi = \text{relative tolerance}$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time</th>
<th>Iterations</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\phi=0.01$</td>
<td>$\phi=0.001$</td>
<td>$\phi=0.01$</td>
</tr>
<tr>
<td>80bau3b</td>
<td>1220</td>
<td>1077</td>
<td>32059</td>
</tr>
<tr>
<td>stocfor2</td>
<td>99</td>
<td>652</td>
<td>3899</td>
</tr>
<tr>
<td>degen3</td>
<td>79</td>
<td>4355</td>
<td>1692</td>
</tr>
<tr>
<td>sctap3</td>
<td>32</td>
<td>156</td>
<td>1145</td>
</tr>
<tr>
<td>pilot</td>
<td>1826</td>
<td>8085</td>
<td>28392</td>
</tr>
<tr>
<td>ganges</td>
<td>175</td>
<td>998</td>
<td>6982</td>
</tr>
<tr>
<td>sierra</td>
<td>90</td>
<td>303</td>
<td>3187</td>
</tr>
<tr>
<td>ship12l</td>
<td>65</td>
<td>524</td>
<td>1436</td>
</tr>
<tr>
<td>ship12s</td>
<td>19</td>
<td>227</td>
<td>638</td>
</tr>
<tr>
<td>woodw</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sctap2</td>
<td>126</td>
<td>808</td>
<td>4516</td>
</tr>
<tr>
<td>scfxm3</td>
<td>147</td>
<td>916</td>
<td>5120</td>
</tr>
<tr>
<td>pilotnov</td>
<td>1207</td>
<td>2099</td>
<td>32423</td>
</tr>
<tr>
<td>pilot_ja</td>
<td>469</td>
<td>3347</td>
<td>11631</td>
</tr>
<tr>
<td>czprob</td>
<td>331</td>
<td>424</td>
<td>8225</td>
</tr>
<tr>
<td>25tv47</td>
<td>198</td>
<td>974</td>
<td>5426</td>
</tr>
<tr>
<td>ship08l</td>
<td>66</td>
<td>298</td>
<td>1379</td>
</tr>
<tr>
<td>ship08s</td>
<td>40</td>
<td>75</td>
<td>1209</td>
</tr>
<tr>
<td>pilot_we</td>
<td>285</td>
<td>1148</td>
<td>7706</td>
</tr>
<tr>
<td>nesm</td>
<td>527</td>
<td>1459</td>
<td>10695</td>
</tr>
<tr>
<td>scfxm2</td>
<td>130</td>
<td>640</td>
<td>4608</td>
</tr>
<tr>
<td>perold</td>
<td>144</td>
<td>1248</td>
<td>4779</td>
</tr>
<tr>
<td>gfrd-pnc</td>
<td>247</td>
<td>1242</td>
<td>10541</td>
</tr>
<tr>
<td>shell</td>
<td>138</td>
<td>376</td>
<td>4935</td>
</tr>
<tr>
<td>fffff800</td>
<td>191</td>
<td>-</td>
<td>5725</td>
</tr>
<tr>
<td>agg2</td>
<td>24</td>
<td>1209</td>
<td>767</td>
</tr>
<tr>
<td>agg3</td>
<td>20</td>
<td>214</td>
<td>660</td>
</tr>
<tr>
<td>seba</td>
<td>117</td>
<td>-</td>
<td>3909</td>
</tr>
<tr>
<td>scrs8</td>
<td>79</td>
<td>1324</td>
<td>2937</td>
</tr>
<tr>
<td>agg</td>
<td>26</td>
<td>61</td>
<td>891</td>
</tr>
<tr>
<td>scagr25</td>
<td>46</td>
<td>76</td>
<td>2034</td>
</tr>
<tr>
<td>standmps</td>
<td>63</td>
<td>512</td>
<td>2200</td>
</tr>
<tr>
<td>grow22</td>
<td>43</td>
<td>44</td>
<td>1017</td>
</tr>
<tr>
<td>pilot4</td>
<td>124</td>
<td>479</td>
<td>3650</td>
</tr>
<tr>
<td>ship04l</td>
<td>69</td>
<td>208</td>
<td>1526</td>
</tr>
<tr>
<td>ship04s</td>
<td>49</td>
<td>83</td>
<td>1411</td>
</tr>
<tr>
<td>etamacro</td>
<td>34</td>
<td>-</td>
<td>1417</td>
</tr>
<tr>
<td>Problem</td>
<td>Time $\phi=0.01$</td>
<td>Iterations $\phi=0.01$</td>
<td>Error $\phi=0.01$</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>scsd8</td>
<td>456</td>
<td>839</td>
<td>.01449</td>
</tr>
<tr>
<td>scorpion</td>
<td>19</td>
<td>839</td>
<td>.0294</td>
</tr>
<tr>
<td>standgub</td>
<td>57</td>
<td>1919</td>
<td>.00807</td>
</tr>
<tr>
<td>standata</td>
<td>56</td>
<td>1919</td>
<td>.00807</td>
</tr>
<tr>
<td>stair</td>
<td>64</td>
<td>2067</td>
<td>.00359</td>
</tr>
<tr>
<td>tuff</td>
<td>1328</td>
<td>-</td>
<td>.77668</td>
</tr>
<tr>
<td>scfxm1</td>
<td>110</td>
<td>4081</td>
<td>.02000</td>
</tr>
<tr>
<td>bandm</td>
<td>42</td>
<td>1549</td>
<td>.0192</td>
</tr>
<tr>
<td>sctapl1</td>
<td>37</td>
<td>1470</td>
<td>.00668</td>
</tr>
<tr>
<td>grow15</td>
<td>24</td>
<td>586</td>
<td>.00074</td>
</tr>
<tr>
<td>capri</td>
<td>111</td>
<td>4493</td>
<td>.04631</td>
</tr>
<tr>
<td>bore3d</td>
<td>14</td>
<td>590</td>
<td>.00399</td>
</tr>
<tr>
<td>e226</td>
<td>79</td>
<td>2370</td>
<td>.01299</td>
</tr>
<tr>
<td>brandy</td>
<td>47</td>
<td>1687</td>
<td>.02051</td>
</tr>
<tr>
<td>sc205</td>
<td>99</td>
<td>4920</td>
<td>.99251</td>
</tr>
<tr>
<td>vtp.base</td>
<td>111</td>
<td>5010</td>
<td>.1514</td>
</tr>
<tr>
<td>israel</td>
<td>17</td>
<td>458</td>
<td>.00319</td>
</tr>
<tr>
<td>beaconfd</td>
<td>80</td>
<td>1882</td>
<td>.00377</td>
</tr>
<tr>
<td>forplan</td>
<td>284</td>
<td>5774</td>
<td>.41775</td>
</tr>
<tr>
<td>scsd6</td>
<td>41</td>
<td>616</td>
<td>.00368</td>
</tr>
<tr>
<td>grow7</td>
<td>24</td>
<td>644</td>
<td>.00799</td>
</tr>
<tr>
<td>scagr7</td>
<td>37</td>
<td>1727</td>
<td>.02486</td>
</tr>
<tr>
<td>stocfor1</td>
<td>19</td>
<td>879</td>
<td>.00233</td>
</tr>
<tr>
<td>share1b</td>
<td>169</td>
<td>5986</td>
<td>.03972</td>
</tr>
<tr>
<td>sc105</td>
<td>34</td>
<td>1620</td>
<td>.04351</td>
</tr>
<tr>
<td>share2b</td>
<td>26</td>
<td>932</td>
<td>.28628</td>
</tr>
<tr>
<td>recipe</td>
<td>10</td>
<td>401</td>
<td>.00420</td>
</tr>
<tr>
<td>scsd1</td>
<td>73</td>
<td>1182</td>
<td>.00360</td>
</tr>
<tr>
<td>adlittle</td>
<td>17</td>
<td>672</td>
<td>.00418</td>
</tr>
<tr>
<td>sc50a</td>
<td>20</td>
<td>941</td>
<td>.01075</td>
</tr>
<tr>
<td>afiro</td>
<td>40</td>
<td>1973</td>
<td>.00184</td>
</tr>
</tbody>
</table>

Average | 181 | 849 | 4906 | 23100 |

Table 5: Parallel run time (msec), number of iterations and relative error in the objective function value for Jacobi base case; $\phi =$ relative tolerance.
Unconstrained Line Search

First, consider the two stepsize variants, where for primal updates, an unconstrained line search is performed, thereafter an under-relaxed step towards that optimum is taken and finally projection onto the simple bounds is carried out. Table 6 shows the relative performance of these variants against each other. (Here the constant stepsize and unconstrained line search refers to the Jacobi base case.) The figures are the average speedup against Minos 5.1, the average relative speedup against Jacobi base case and the number of non-convergent problems. We may observe that on the average the safe steps result in 26% decrease in efficiency. The reason for this is not obvious, given that line search is applied in each case.

Constrained Line Search

Next, consider variants, where the simple bounds are taken into account in the line search optimization. Table 6 shows the performance of the resulting two variants as well. In both cases the solution efficiency decreased considerably as compared with the
variants employing unconstrained line search. For constant stepsize, for instance, the average run time about doubled.

No Line Search

Finally, Table 6 shows the result, when line search is omitted so that the primal update is obtained directly employing the stepsize and projection. These variants appear somewhat less robust (in comparison with unconstrained line search) in that more nonconvergent problems appear, and the run time increases as well.

7.3 Parameter Sensitivity

Sensitivity tests concerning program parameters were carried out by varying the default values one at a time. Again these tests are performed with the relative tolerance $\phi = 0.01$ and using 50 smallest of the problems in Tables 1 and 2 only. The results are summarized in Table 7 showing the average speedup against Minos 5.1, the average relative speedup against Jacobi base case and the number of non-convergent problems. Note that the speedups are defined as averages over the problems which converged in the particular case. Thus it may happen that the speedup against Minos is larger than in the base case, yet the relative speedup is less than 1. (see $\rho = 1.$, for instance, where two nonconvergent problems appear). The default values of parameters are indicated in parentheses.

We conclude that a line search stepsize $\omega = 1.0$ results in a run time increase of 60% as compared with the Jacobi base case. This is explained by often too long steps which after projection result in a decrease in the function value. On the other hand, $\omega = 0.3$ seems too conservative, so that on the average the run time increases by about 40%. For the penalty parameter, $\rho = 0.2$ appears too small resulting in slow convergence of the dual multipliers, and $\rho = 1.0$ seems too large so that the convergence of the primal variables slows down. Besides, two nonconvergencies appear in the latter case. The run time sensitivity with respect to changes in $\alpha$ appears relatively small.

For dual stepsize $\gamma$, the default value $\gamma = 0.1$ performs better than comparing values $\gamma = 0.05$ and $\gamma = 0.2$. The loss in efficiency in these two cases is 30-50%. A similar conclusion applies to the dual update threshold parameter $\beta$ for which the default value is $\beta = 1$. The values $\beta = 0.5$ and $\beta = 2$ resulted in a loss of efficiency of 20-30%. For $\beta = 0$, the run time tripled on the average and two problems failed to convergence. This may justify the use of parameter $\beta$.

Changes up and down in the primal stepsize constant $\tau$ results in some loss of efficiency, in particular for the smaller value $\tau = 0.05$, where robustness suffers as well. For the primal update threshold $\eta$, instead, a change down from the default value $\eta = 1$ always improves. Besides, generally the precision in the objective function value improves as well. These runs may be considered as our best cases of Jacobi so far.

As a final parameter test we experimented with multiple sweeps of primal and dual phases between the updates of the dual multipliers $\tau_i$. The approach is the same as the Jacobi base case except that we update the dual multipliers in every $\kappa$th iteration only. We experimented with $\kappa$ equal to 2, 5, 10 and 20, and $\gamma$ equal to 0.1, 0.2, 0.5,
<table>
<thead>
<tr>
<th>Parameter Configuration</th>
<th>Speedup</th>
<th>Relative Speedup</th>
<th>Non-Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi Base Case</td>
<td>37.7</td>
<td>1.00</td>
<td>0</td>
</tr>
<tr>
<td>Line search stepsize ($\omega = 0.5$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\omega = 0.3$</td>
<td>34.2</td>
<td>0.71</td>
<td>1</td>
</tr>
<tr>
<td>$\omega = 1.0$</td>
<td>23.2</td>
<td>0.64</td>
<td>1</td>
</tr>
<tr>
<td>Penalty parameter ($\rho = 0.5$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho = 0.2$</td>
<td>29.9</td>
<td>0.79</td>
<td>0</td>
</tr>
<tr>
<td>$\rho = 1.0$</td>
<td>41.9</td>
<td>0.86</td>
<td>2</td>
</tr>
<tr>
<td>Exponential smoothing ($\alpha = 0.5$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha = 0.3$</td>
<td>47.7</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha = 0.7$</td>
<td>32.2</td>
<td>0.85</td>
<td>0</td>
</tr>
<tr>
<td>Dual stepsize ($\gamma = 0.10$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\gamma = 0.05$</td>
<td>32.0</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma = 0.20$</td>
<td>37.1</td>
<td>0.78</td>
<td>1</td>
</tr>
<tr>
<td>Dual update threshold ($\beta = 1.$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta = 0.0$</td>
<td>15.7</td>
<td>0.33</td>
<td>2</td>
</tr>
<tr>
<td>$\beta = 0.5$</td>
<td>31.7</td>
<td>0.84</td>
<td>0</td>
</tr>
<tr>
<td>$\beta = 2.0$</td>
<td>35.7</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>Primal stepsize ($\tau = 0.10$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau = 0.05$</td>
<td>38.3</td>
<td>0.78</td>
<td>2</td>
</tr>
<tr>
<td>$\tau = 0.15$</td>
<td>33.8</td>
<td>0.90</td>
<td>0</td>
</tr>
<tr>
<td>Primal update threshold ($\eta = 1.0$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\eta = 0.0$</td>
<td>46.2</td>
<td>1.23</td>
<td>0</td>
</tr>
<tr>
<td>$\eta = 0.5$</td>
<td>43.6</td>
<td>1.16</td>
<td>0</td>
</tr>
<tr>
<td>$\eta = 0.9$</td>
<td>45.2</td>
<td>1.20</td>
<td>0</td>
</tr>
<tr>
<td>Number of sweeps ($\kappa = 1, \gamma = 0.1$)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\kappa = 5, \gamma = 0.1$</td>
<td>16.0</td>
<td>0.54</td>
<td>1</td>
</tr>
<tr>
<td>$\kappa = 5, \gamma = 0.2$</td>
<td>30.8</td>
<td>0.64</td>
<td>1</td>
</tr>
<tr>
<td>$\kappa = 5, \gamma = 0.5$</td>
<td>32.6</td>
<td>0.86</td>
<td>0</td>
</tr>
<tr>
<td>$\kappa = 5, \gamma = 1.0$</td>
<td>35.8</td>
<td>0.93</td>
<td>1</td>
</tr>
<tr>
<td>$\kappa = 5, \gamma = 1.5$</td>
<td>43.8</td>
<td>1.18</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7: Sensitivity with respect to parameters $\omega$, $\rho$, $\alpha$, $\gamma$, $\beta$, $\tau$ and $\eta$ (default values are indicated in parentheses). The figures are average speed-up against Minos 5.1, average relative speed-up against Jacobi base case and the number of nonconvergent problems.
1.0 and 1.5 (all other parameters having their default values). For \( k = 2 \), in the best case with \( \gamma = 0.2 \), the parallel time was 9% less than in the Jacobi base case. For other values of \( \gamma \) the performance slightly deteriorated as compared with the base case. For \( k = 5 \), the results are given in Table 7. The only case with an increase in the average speed-up occurs with \( \gamma = 1.5 \); however, two nonconvergencies appeared so that the case is less robust compared with the Jacobi base case. For \( k \) equal to 10 and 20, improvement with respect to the Jacobi base case was not achieved, neither in speed-up nor in robustness. In conclusion, we observe that an increase in the number of sweeps \( k \) allows an increase in stepsize \( \gamma \) for the dual multipliers. However, at the same time the robustness of the approach tends to suffer without any significant gain in the run time.

7.4 Comparison with Gauss-Seidel and Interior Point Methods

Figure 11 shows the speed-up of the base case of Jacobi against a serial algorithm based on Gauss-Seidel type of iterations for minimizing augmented Lagrangians [14]. The problems from Tables 1 and 2 for which both methods converged were chosen for comparisons. The Gauss-Seidel approach basically differs from Jacobi in that the primal variables are updated sequentially, and the price vector \( p \) is revised after the update of each primal variable. No line search is employed in Gauss-Seidel. Again the Minos time is taken as a measure of problem complexity. We may conclude from Figure 11, that the speed-ups appear somewhat independent of the problem complexity. However, when the relative tolerance \( \phi \) is decreased from 0.01 to 0.001, the average speedup decreases from 26 to 9, on the average.

Finally, Table 8 shows a comparison between Karmarkar's algorithm, Jacobi base case and Minos 4.0. The speed-up factors for Karmarkar are adopted from [1], from which all Netlib problems with over 800 constraints have been chosen for the comparison. Again, the comparison is made assuming that Karmarkar's algorithm and Minos are run in a serial and Jacobi in a parallel computer. For relative tolerance \( \phi = 0.01 \) and \( \phi = 0.001 \), the speed-ups of Jacobi appear favorable. However, for \( \phi = 0.01 \), the precision is not always satisfactory, notably for the problem 25fv47 (see Table 4).

8. Conclusions

The linear programming method analysed in this paper updates the dual variables as in the multiplier method and the primal variables by under-relaxed Jacobi-type iterations. Each iteration of the method is highly parallelizable.

Our theoretical analysis provides us with the estimates of safe stepsize coefficients and of the convergence ratio of the method. They turn out to be independent on the size of the problem and on penalty value and are determined mainly by the average number of nonzeros that appear in the rows of the constraint matrix.

The computational experience gained so far indicates that the method can be very efficient for large problems, especially in parallel computing environments, where we
Table 8: Speed-up factors for Karmarkar’s algorithm and Jacobi base case relative to Minos 4.0; $\phi = \text{relative tolerance}$.

can expect speedups of many orders of magnitude. It also shows that our theoretical estimates of safe stepsizes have practical importance.

Acknowledgement

Financial support for this work is gratefully acknowledged from The Foundation for the Helsinki School of Economics. The authors also thank Ms. Irina Ivanova for her help in preparing the pictures.
References


Figure 1. Speed-up of *Jacobi* base case against Minos 5.1 as a function of serial run time for Minos; $\phi = \text{relative tolerance}$. 
Figure 2. Speed-up Jacobi base case against Minos 5.1 as a function of the inverse of problem density; $\phi =$ relative tolerance.

$\phi = .01$

$\phi = .001$
Figure 3. Parallel run time (sec) for Jacobi base case; $\phi$ = relative tolerance.

$\phi = 0.01$

$\phi = 0.001$
Figure 4. Number of iterations for *Jacobi* base case; $\phi = \text{relative tolerance}$. 

\[ \phi = .01 \]

\[ \phi = .001 \]
Figure 5. Relative error in the objective function value for Jacobi base case; $\phi =$ relative tolerance.
Figure 6. The run for problem 80bau3b: Relative error of the objective function value as a function of iterations.
Figure 7. The run for problem 80bau3b: Maximal relative primal and dual error as a function of iterations.
Figure 8. The run for problem \textit{80kau3b}: The number of primal and dual infeasibilities as a function of iterations.
Figure 9. The run for problem 80bau3b: Minimum, maximum and average value of the safe stepsize $\tau_j$ as a function of iterations.
Figure 10. The run for problem 80bau3b: The number of active rows and columns (in percent relative to m) as a function of iterations.
Figure 11. Speed-up of *Jacobi* base case against Gauss-Seidel in a serial computer; $\phi =$ relative tolerance.

\[ \phi = 0.01 \]

\[ \phi = 0.001 \]