International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

A Regularized Jacobi Method for Large-Scale Linear Programming

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

IIASA Working Paper

November 1993

Working Paper

A regularized Jacobi method for large-scale linear programming

Markku Kallio
Andrzej Ruszczyński
Seppo Salo

WP-93-61
November 1993

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 715210 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

A regularized Jacobi method for large-scale linear programming

Markku Kallio
Andrzej Ruszczyński
Seppo Salo

WP-93-61
November 1993

International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 715210 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

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

$$\begin{aligned} &\min c^T x \\ &Ax = b \qquad (1.1) \\ &l \leq x \leq u, \end{aligned}$$

where $x \in R^n$ is the vector of decision variables, c, l and u are some vectors in R^n, $b \in R^m$ 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 $\pi \in R^m$ is the vector of multipliers associated with the equality constraint $Ax = b$. The Lagrangian can be used to define the dual function

$$\hat{L}(\pi) = \inf_{l \le x \le u} L(x, \pi) = b^T \pi + \sum_{j=1}^{n} \hat{L}_j(\pi) \tag{2.1}$$

with

$$\hat{L}_j(\pi) = \inf_{l_j \le x_j \le u_j} (c_j - \pi^T A_j) x_j, \tag{2.2}$$

where A_j denotes the jth column of A.

The dual problem associated with (1.1) is formulated as

$$\max_{\pi \in R^m} \hat{L}(\pi). \tag{2.3}$$

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 $l \le \hat{x} \le u$ is an optimal solution of (1.1) if and only if

$$(c_j - \hat{\pi}^T A_j)\hat{x}_j = \min_{l_j \le x_j \le u_j} (c_j - \hat{\pi}^T A_j)x_j, \quad j = 1, \ldots, n \tag{2.4}$$

$$A\hat{x} = b. \tag{2.5}$$

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:

$$\Lambda(x, \pi) = c^T x + \pi^T(b - Ax) + \frac{1}{2}\rho\|b - Ax\|^2, \tag{2.6}$$

where ρ is a positive penalty parameter. Similarly to (2.1)-(2.3) we can introduce the regularized dual function

$$\hat{\Lambda}(\pi) = \inf_{l \le x \le u} \Lambda(x, \pi), \tag{2.7}$$

the set of minimizers

$$\hat{X}(\pi) = \{l \le x \le u : \ \Lambda(x, \pi) = \hat{\Lambda}(\pi)\} \tag{2.8}$$

and the regularized dual problem

$$\max_{\pi \in R^m} \hat{\Lambda}(\pi). \tag{2.9}$$

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)

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

(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

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

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

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

Then (2.10) can be rewritten component-wise as

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

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 π^k find a solution x^k of the problem

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

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

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

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 jth variable and assume that all other variables x_s, $s \neq j$ are fixed at some values $x_s^{k,\nu}$, the best value of x_j can be found from the problem

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

This can be deciphered as

$$\min_{l_j \leq x_j \leq u_j} \tilde{\Lambda}_j(x_j, x^{k,\nu}, \pi^k) \tag{3.1}$$

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 $y^{k,\nu}$ and current prices $p^{k,\nu}$ by

$$y^{k,\nu} = b - Ax^{k,\nu}, \tag{3.2}$$

$$p^{k,\nu} = \pi^k + \rho y^{k,\nu}. \tag{3.3}$$

Primal Phase. Calculate the reduced cost

$$\bar{c}^{k,\nu} = c - A^T p^{k,\nu}. \tag{3.4}$$

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

$$\xi_j^{k,\nu} = \left[x_j^{k,\nu} - \frac{\bar{c}_j^{k,\nu}}{\rho \|A_j\|^2} \right]_{l_j}^{u_j}, \tag{3.5}$$

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

$$x_j^{k,\nu+1} = x_j^{k,\nu} + \tau_j^{k,\nu}(\xi_j^{k,\nu} - x_j^{k,\nu}), \tag{3.6}$$

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

Stopping Test. If $\xi^{k,\nu} \neq x^{k,\nu}$ then increase ν by one and go to Dual Phase; otherwise, if $y^{k,\nu} \neq 0$, then set

$$\pi^{k+1} = \gamma p^{k,\nu} + (1 - \gamma)\pi^k, \tag{3.7}$$

$x^k = x^{k,\nu}$, increase k 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 τ_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 ith row

of A, $i = 1,\ldots,m$. We define for each column A_j of A, $j = 1,\ldots,n$, the average rowcount

$$N_j = \frac{\sum_{i=1}^m a_{ij}^2 M_i}{\sum_{i=1}^m a_{ij}^2}. \tag{4.1}$$

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 $d \in R^n$ and every $h \in R^n$*

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

Proof. Expansion into single entries yields

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

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

$$\begin{aligned}(\langle Ad, Ah\rangle)^2 &\leq \left(\sum_{i=1}^m \sum_{j=1}^n \sum_{s\in V(i,j)} a_{ij}^2 d_j^2\right)\left(\sum_{i=1}^m \sum_{j=1}^n \sum_{s\in V(i,j)} a_{ij}^2 h_j^2\right) \\ &= \left(\sum_{i=1}^m \sum_{j=1}^n M_i a_{ij}^2 d_j^2\right)\left(\sum_{i=1}^m \sum_{j=1}^n M_i a_{ij}^2 h_j^2\right),\end{aligned}$$

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

$$\xi_j^{k,\nu} = x_j^{k,\nu} - \zeta_j^{k,\nu} d_j^{k,\nu}, \tag{4.2}$$

where $d_j^{k,\nu}$ is the unconstrained step given by

$$d_j^{k,\nu} = \frac{\bar{c}_j^{k,\nu}}{\rho \|A_j\|^2} \tag{4.3}$$

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*

$$0 < \tau_j^{k,\nu} < \frac{2}{\zeta_j^{k,\nu} N_j}, \quad j = 1, \ldots, n, \tag{4.4}$$

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 $\left\{x^{k,\nu}\right\}_{\nu=0}^{\infty}$ is a solution of (2.12).

Proof. Let us denote

$$\tau = \begin{bmatrix} \tau_1^{k,\nu} & & & \\ & \tau_2^{k,\nu} & & \\ & & \ddots & \\ & & & \tau_n^{k,\nu} \end{bmatrix}.$$

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

$$\Lambda(x + \tau(\xi - x), \pi^k) - \Lambda(x, \pi^k) \;=\; \bar{c}^T \tau(\xi - x) + \frac{1}{2}\rho \|A\tau(\xi - x)\|^2.$$

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

$$\begin{aligned} &\Lambda(x + \tau(\xi - x), \pi^k) - \Lambda(x, \pi^k) \leq \\ &\quad \sum_{j=1}^{n} \tau_j \bar{c}_j (\xi_j - x_j) + \frac{1}{2}\rho \sum_{j=1}^{n} \tau_j^2 N_j \|A_j\|^2 (\xi_j - x_j)^2. \end{aligned} \tag{4.5}$$

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

$$\Lambda(x^{k,\nu+1}, \pi^k) - \Lambda(x^{k,\nu}, \pi^k) \leq -\rho \sum_{j=1}^{n} \tau_j^{k,\nu} \left(1 - \frac{\tau_j^{k,\nu} \zeta_j^{k,\nu} N_j}{2}\right) \|A_j\|^2 \zeta_j^{k,\nu} (d_j^{k,\nu})^2. \tag{4.6}$$

Thus for τ_j's satisfying (4.4) the sequence $\left\{\Lambda(x^{k,\nu}, \pi^k)\right\}_{\nu=0}^{\infty}$ 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 $\xi_j^{k,\nu}$,

$$\xi_j^{k,\nu} = \left[\xi_j^{k,\nu} - \frac{\partial\tilde{\Lambda}_j(\xi_j^{k,\nu}, x^{k,\nu}, \pi^k)}{\partial\xi_j}\right]_{l_j}^{u_j}.$$

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

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

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 $\zeta_j^{k,\nu}$ we suppress the iteration counter ν 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

$$\hat{c} = c - A^T\pi - \rho A^T(b - A\hat{x}) = \hat{\eta} - \hat{\xi}, \tag{5.1}$$

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^ν is

$$\bar{c} = c - A^T\pi - \rho A^T(b - Ax^\nu) = \hat{\eta} - \hat{\xi} + \rho A^T A(x^\nu - \hat{x}). \tag{5.2}$$

For the optimality gap we obtain by direct calculation

$$\begin{aligned}
\Delta_\nu &= \Lambda(x^\nu, \pi) - \hat{\Lambda}(\pi) \\
&= (c - A^T(\pi + \rho b))^T(x^\nu - \hat{x}) + \frac{\rho}{2} \parallel Ax^\nu \parallel^2 - \frac{\rho}{2} \parallel A\hat{x} \parallel^2 \\
&= \frac{\rho}{2} \parallel A(x^\nu - \hat{x}) \parallel^2 + \hat{\eta}^T(x^\nu - l) + \hat{\xi}^T(u - x^\nu).
\end{aligned} \tag{5.3}$$

Lemma 2. *If the sequence $\Delta_\nu, \nu = 1, 2, ...,$ converges to zero, then there exists $\bar{\nu}$ such that, for all $\nu \geq \bar{\nu}$, there is $\hat{x}^\nu \in \hat{X}(\pi)$ with the property $\hat{x}_j^\nu = x_j^\nu$, for all j such that $x_j^\nu = l_j$ or $x_j^\nu = u_j$.*

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

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

$$\tau_j = min\,(1, \mu/(\zeta_j N_j))\,.$$

Then the Regularized Jacobi Method converges linearly:

$$\Delta_{\nu+1} \leq q\Delta_\nu \tag{5.4}$$

with the convergence ratio q such that

$$q \leq \left(1 - \frac{\mu^* \lambda}{\theta}\right)^2 < 1, \tag{5.5}$$

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

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

and

$$\mu^* = \min\,(\mu, 1 - \mu/\alpha)\,, \tag{5.6}$$

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

$$2 - \alpha = \frac{\lambda \mu^*}{\theta}. \tag{5.7}$$

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, $\Delta_\nu \to 0$, so that by (5.3) $A(x^\nu - \hat{x}) \to 0, \hat{\eta}^T(x^\nu - l) \to 0$ and $\hat{\xi}^T(u - x^\nu) \to 0$. Therefore, $\bar{c} \to \hat{\eta} - \hat{\xi}$ in (5.2), $x_j^\nu \to l_j$, for all $j \in J_L$, $x_j^\nu \to u_j$, for all $j \in J_U$. Therefore there exists $\bar{\nu}$ such that, for all $\nu \geq \bar{\nu}$ and $j \in J_L \cup J_U$, (employing (3.4), (4.2) and (4.4)), τ_j is given as

$$\tau_j = \min\left[1, \frac{\mu}{\zeta_j N_j}\right] = \min\left[1, \frac{\mu}{N_j} \max\left(1, \frac{-\bar{c}_j}{\rho \|A_j\|^2 (\hat{x}_j - x_j^\nu)}\right)\right] = 1.$$

Note that $\tau_j = 1$ implies that the componenent x_j ends up on its bound. Thus, for all $\nu \geq \bar{\nu}$, $x_j^\nu = \hat{x}_j$, 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 as study the iteration from x^ν 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 α 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 μ^* given by (5.6), we obtain

$$\mu^*/N_j \leq \tau_j \zeta_j \leq \mu/N_j, \ j \in J_B, \tag{5.8}$$

$$0 \leq \tau_j \zeta_j \leq (1 - \mu/\alpha)/N_j, \ j \in J_N. \tag{5.9}$$

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 τ_B, ζ_B and S_B be diagonal matrices with entries τ_j, ζ_j and $\|A_j\|$, for $j \in J_B$, respectively. Similarly for J_N we define diagonal matrices τ_N, ζ_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),

$$\zeta_j \bar{c}_j = \zeta_j \rho A_j^T A(x^\nu - \hat{x}),$$

whence

$$\begin{aligned} x_j^{\nu+1} - x_j^\nu &= -\frac{1}{\rho} S_j^{-2} \tau_j \zeta_j \bar{c}_j \\ &= -S_j^{-2} \tau_j \zeta_j A_j^T A(x^\nu - \hat{x}). \end{aligned} \tag{5.10}$$

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

$$\begin{aligned} \bar{B} &= B S_B^{-1} \tau_B^{1/2} \zeta_B^{1/2}, \\ \bar{N} &= N S_N^{-1} \tau_N^{1/2} \zeta_N^{1/2}. \end{aligned}$$

Let us define the matrix

$$E = (I - \bar{N}\bar{N}^T).$$

From (5.12) we obtain

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

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

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

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

The optimality gap in (5.3) simplifies into

$$\Delta_\nu = \frac{\rho}{2}\|e^\nu\|^2, \tag{5.15}$$

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

$$e^\nu = E^{-1} e_B^\nu = E^{-1}\bar{B}u, \tag{5.16}$$

with

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

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

$$e^{\nu+1} = (E - \bar{B}\bar{B}^T)E^{-1}\bar{B}u. \tag{5.17}$$

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

$$\begin{aligned} \frac{\Delta_{\nu+1}}{\Delta_\nu} &= \frac{\|E^{1/2}(I - QQ^T)Qu\|^2}{\|E^{-1/2}Qu\|^2} \\ &\leq \bar{\lambda}_E^2 \frac{\|(I - QQ^T)Qu\|^2}{\|Qu\|^2} \\ &\leq \bar{\lambda}_E^2 \max_t (1 - \lambda_t)^2, \end{aligned} \tag{5.18}$$

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

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

with $\lambda_Q^{\min} = \min_t \lambda_t$.

Let $\lambda_{\bar{B}}^{\min}$ and $\lambda_B^{\min}$ denote the smallest positive eigenvalue of $\bar{B}\bar{B}^T$ and BB^T, respectively. Then $\lambda_Q^{\min} \geq \lambda_{\bar{B}}^{\min}/\bar{\lambda}_E$. Consequently,

$$\bar{\lambda}_E^2\left(1 - \lambda_Q^{\min}\right)^2 \leq (\bar{\lambda}_E - \lambda_{\bar{B}}^{\min})^2. \tag{5.20}$$

With μ^* and θ given by (5.6), employing (5.8) we have

$$\mu \geq \bar{\lambda}_{\bar{B}} \geq \lambda_{\bar{B}}^{\min} \geq \lambda_B^{\min} \min_j (S_j^{-2} \tau_j \zeta_j) \geq \frac{\mu^* \lambda_B^{\min}}{\theta} \geq \frac{\mu^* \lambda}{\theta} > 0. \tag{5.21}$$

In summary, $\bar{\lambda}_E \in [\mu, 1]$ and $\lambda_{\bar{B}}^{\min} \in [\mu^* \lambda/\theta, \mu]$. 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 α^* 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 θ in (5.5) equals just $\max_j N_j$, while λ 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 $\bar{\nu}$ be defined as in the proof of theorem 2. For $\nu \geq \bar{\nu}$ 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

$$\|x^\nu - \hat{x}\| \leq C_1 \sum_{l=\nu}^{\infty} q^{l/2} = \frac{C_1 q^{\nu/2}}{1 - q^{1/2}}.$$

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 τ_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 unnesessary, 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 $\delta = (\delta_j) \in R^n$ depending on the current solution will be applied while checking dual feasibility and complementarity: a reduced cost $\bar{c}_j$ vanishes, if it is equal to zero within tolerance δ_j. Similarly, a positive primal tolerance vector $\epsilon = (\epsilon_i) \in R^m$ depending on the current solution is employed to test

primal feasibility. Tolerances δ and ϵ are defined for each iteration as follows. First, a reference primal vector $Q = (Q_i) \in R^m$ and a reference dual vector $V = (V_j) \in R^n$ are defined so that

$$\begin{aligned} Q_i &= |b_i| + \sum_{j=1}^{n} |x_j a_{ij}|, \\ V_j &= |c_j| + \sum_{i=1}^{m} |p_i a_{ij}|, \end{aligned}$$

where x_j and p_i denote primal and dual solutions at the beginning of the current iteration. We then define auxiliary tolerance vectors δ' and ϵ' equal to a fraction of reference vectors V and Q, respectively:

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

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

$$\begin{aligned} \delta_j &= \max(\sigma, \alpha\delta'_j + (1-\alpha)\delta_j^-), \\ \epsilon_j &= \max(\sigma, \alpha\epsilon'_j + (1-\alpha)\epsilon_j^-), \end{aligned}$$

where α 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 π. If the resulting optimum price vector is p, then the primal error $y = D^{-1}(p - \pi)$ tends to meet the tolerance ϵ. 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 π_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 π_i by $\gamma_i y_i D_i$ (cf. (2.13)), where γ_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 ρS_i. Here $\sqrt{S_i}$ serves as a scaling factor for row i and the penalty factor ρ is the penalty parameter in (2.6) (applied to the problem with rows scaled by $\sqrt{S_i}$). Again, let δ_j and ϵ_i be the dual and primal tolerances. The penalty parameter is then defined following [14] :

$$D_i = \rho E_j \left\{ \frac{\delta_j}{|a_{ij}|\epsilon_i} \right\}, \tag{6.1}$$

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 α 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 π_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 π according to (3.7) with under-relaxation. The under-relaxation parameter γ is constant over the iterations, unless the relative primal error $\mu_i = |y_i|/\epsilon_i < \beta$ where β is a dual update threshold. In the latter case, γ is scaled down by the factor μ_i/β (the significance of this detail is tested in section 7.3). For inequality rows, π_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 $A_j^T D A_j$.

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 τ_j a stepsize parameter. We define

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

   where ζ_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 η 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 τ_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 ω. 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 π_i 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 ϕ, 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 π_i 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 θm, 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 λ 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 λ 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:

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

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

Problem	Rows	Columns	Nonzeros	Time
80bau3b	2263	9799	29063	272.9
stocfor2	2158	2031	9492	37.1
degen3	1504	1818	26230	147.9
sctap3	1481	2480	10734	11.6
pilot	1442	3652	43220	1153.0
ganges	1310	1681	7021	7.5
sierra	1228	2036	9252	13.6
ship12l	1152	5427	21597	12.7
ship12s	1152	2763	10941	5.2
woodw	1099	8405	37478	89.0
sctap2	1091	1880	8124	7.2
scfxm3	991	1371	7846	12.9
pilotnov	976	2172	13129	32.9
pilot_ja	941	1988	14706	105.5
czprob	930	3523	14173	16.9
25fv47	822	1571	11127	89.6
ship08l	779	4283	17085	4.7
ship08s	779	2387	9501	2.1
pilot_we	723	2789	9218	56.2
nesm	663	2923	13988	27.6
scfxm2	661	914	5229	6.3
perold	626	1376	6026	32.3
gfrd-pnc	617	1092	3467	3.6
shell	537	1775	4900	1.66
fffff800	525	854	6235	6.08
agg2	517	302	4515	1.16
agg3	517	302	4531	1.22
seba	516	1028	4874	1.96
scrs8	491	1169	4029	3.08
agg	489	163	2541	.84
scagr25	472	500	2029	1.86
standmps	468	1075	3686	1.20
grow22	441	946	8318	7.80
pilot4	411	1000	5145	11.08
ship04l	403	2118	8450	1.32
ship04s	403	1458	5810	.66
etamacro	401	688	2489	1.70

Table 1: Number of rows, columns and non-zeros for Netlib test problems. Time is the reference time (sec) obtained by Minos 5.1.

Problem	Rows	Columns	Nonzeros	Time
scsd8	398	2750	11334	25.86
scorpion	389	358	1708	.36
standgub	362	1184	3147	.36
standata	360	1075	3038	.36
stair	357	467	3857	3.66
tuff	334	587	4523	4.44
scfxm1	331	457	2612	1.32
bandm	306	472	2659	1.92
sctap1	301	480	2052	.62
grow15	301	645	5665	3.74
capri	272	353	1786	.58
bore3d	234	315	1525	.46
e226	224	282	2767	1.26
brandy	221	249	2150	1.38
sc205	206	203	552	.26
vtp.base	199	203	914	.12
israel	175	142	2358	.70
beaconfd	174	262	3476	.26
forplan	162	421	4916	.96
scsd6	148	1350	5666	3.24
grow7	141	301	2633	.76
scagr7	130	140	553	.14
stocfor1	118	111	474	.10
share1b	118	225	1182	.50
sc105	106	103	281	.10
share2b	97	79	730	.12
recipe	92	180	752	.04
scsd1	78	760	3148	.68
adlittle	57	97	465	.06
sc50a	51	48	131	.02
afiro	28	32	88	.004

Table 2: Number of rows, columns and non-zeros for Netlib test problems. Time is the reference time (sec) obtained by Minos 5.1.

Parameter	Default Value	Interpretation
ρ	0.5	Penalty factor
ω	0.5	Line search under-relaxation
α	0.5	Exponential smoothing weight
τ	0.1	Primal stepsize
η	1.0	Primal update threshold
γ	0.1	Dual stepsize
β	1.0	Dual update threshold

Table 3: Default values of parameters

count). 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 ϕ. 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 τ_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 × 4585 and 5171 × 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

Problem	Time		Iterations		Error	
	ϕ=0.01	ϕ=0.001	ϕ=0.01	ϕ=0.001	ϕ=0.01	ϕ=0.001
80bau3b	1220	1077	32059	28591	.00112	.00056
stocfor2	99	652	3899	25349	.00573	.00007
degen3	79	4355	1692	103296	.00220	.00010
sctap3	32	156	1145	5392	.00185	.00003
pilot	1826	8085	28392	136015	.08303	.00166
ganges	175	998	6982	39307	.24595	.01750
sierra	90	303	3187	10490	.00793	.00239
ship12l	65	524	1436	11475	.03603	.00039
ship12s	19	227	638	7540	.02641	.00047
woodw	-	-	-	-	.00596	.00061
sctap2	126	808	4516	30247	.00026	.00000
scfxm3	147	916	5120	32926	.00147	.00009
pilotnov	1207	2099	32423	55711	.00088	.00061
pilot_ja	469	3347	11631	87463	.00662	.00081
czprob	331	424	8225	10312	.00137	.00014
25fv47	198	974	5426	27283	.23585	.00276
ship08l	66	298	1379	6027	.00683	.00012
ship08s	40	75	1209	2275	.00369	.00150
pilot_we	285	1148	7706	31732	.00314	.00055
nesm	527	1459	10695	29481	.00761	.00023
scfxm2	130	640	4608	22929	.00429	.00056
perold	144	1248	4779	41639	.05544	.00043
gfrd-pnc	247	1242	10541	54275	.05750	.00116
shell	138	376	4935	13783	.00003	.00026
fffff800	191	-	5725	-	.00566	.00036
agg2	24	1209	767	41982	.00254	.00015
agg3	20	214	660	6891	.00673	.00067
seba	117	-	3909	-	.02721	.02742
scrs8	79	1324	2937	53044	.01201	.00161
agg	26	61	891	2113	.00750	.00684
scagr25	46	76	2034	3386	.01333	.00139
standmps	63	512	2200	19498	.00607	.00013
grow22	43	44	1017	1039	.00097	.00050
pilot4	124	479	3650	14721	.00686	.00034
ship04l	69	208	1526	4702	.00490	.00031
ship04s	49	83	1411	2398	.00446	.00042
etamacro	34	-	1417	-	.00044	.00002

Table 4: Parallel run time (msec), number of iterations and relative error in the objective function value for *Jacobi* base case; ϕ = relative tolerance.

	Time		Iterations		Error	
Problem	ϕ=0.01	ϕ=0.001	ϕ=0.01	ϕ=0.001	ϕ=0.01	ϕ=0.001
scsd8	456	1894	8396	36819	.01449	.00050
scorpion	19	260	839	11691	.00294	.00032
standgub	57	427	1919	16058	.00807	.00005
standata	56	446	1919	17199	.00807	.00006
stair	64	563	2067	18419	.00359	.00016
tuff	1328	-	40345	-	.77668	.69926
scfxm1	110	234	4081	8906	.00200	.00091
bandm	42	720	1549	27908	.01192	.00385
sctap1	37	239	1470	9825	.00668	.00166
grow15	24	32	586	783	.00074	.00013
capri	111	480	4493	19750	.04631	.00191
bore3d	14	54	590	2225	.00399	.00017
e226	79	1750	2370	59701	.01299	.00172
brandy	47	205	1687	7681	.02051	.00159
sc205	99	167	4920	7919	.99251	.00063
vtp.base	111	120	5010	5376	.01514	.00372
israel	17	152	458	4453	.00319	.00001
beaconfd	80	112	1882	2689	.00377	.00016
forplan	284	880	5774	19072	.41775	.47582
scsd6	41	6146	616	103559	.00368	.00006
grow7	24	48	644	1221	.00799	.00092
scagr7	37	93	1727	4230	.02486	.00039
stocfor1	19	34	879	1552	.00233	.00009
share1b	169	710	5986	26151	.03972	.00212
sc105	34	190	1620	9235	.04351	.00011
share2b	26	1319	932	50768	.28628	.00135
recipe	10	28	401	1119	.00420	.00008
scsd1	73	279	1182	4836	.00360	.00009
adlittle	17	65	672	2461	.00418	.00025
sc50a	20	140	941	6920	.01075	.00111
afiro	40	67	1973	3358	.00184	.00014
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; ϕ = relative tolerance.

Line Search	Stepsize Constant	Safe
	Speedup	
Unconstrained	37.7	35.3
Constrained	18.8	19.0
None	24.0	16.6
	Relative Speedup	
Unconstrained	1.00	.74
Constrained	.49	.47
None	.66	.46
	Non-convergence	
Unconstrained	0	1
Constrained	0	3
None	2	1

Table 6: Average speed-up against Minos 5.1, average relative speed-up against *Jacobi* base case and number of non-convergent problems for two stepsize variants of *Jacobi*.

alternatives: unconstrained, constrained and no line search. We shall carry out tests with the relative tolerance $\phi = 0.01$ and using 50 smallest of the problems in Tables 1 and 2 only.

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 α appears relatively small.

For dual stepsize γ, 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 β 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 β.

Changes up and down in the primal stepsize constant τ 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 η, 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 π_i. The approach is the same as the *Jacobi* base case except that we update the dual multipliers in every κth iteration only. We experimented with κ equal to 2, 5, 10 and 20, and γ equal to 0.1, 0.2, 0.5,

	Speedup	Relative Speedup	Non-Convergence
Jacobi Base Case	37.7	1.00	0
Line search stepsize ($\omega = 0.5$)			
$\omega = 0.3$	34.2	0.71	1
$\omega = 1.0$	23.2	0.64	1
Penalty parameter ($\rho = 0.5$)			
$\rho = 0.2$	29.9	0.79	0
$\rho = 1.0$	41.9	0.86	2
Exponential smoothing ($\alpha = 0.5$)			
$\alpha = 0.3$	47.7	1.00	1
$\alpha = 0.7$	32.2	0.85	0
Dual stepsize ($\gamma = 0.10$)			
$\gamma = 0.05$	32.0	0.67	1
$\gamma = 0.20$	37.1	0.78	1
Dual update threshold ($\beta = 1.$)			
$\beta = 0.0$	15.7	0.33	2
$\beta = 0.5$	31.7	0.84	0
$\beta = 2.0$	35.7	0.75	1
Primal stepsize ($\tau = 0.10$)			
$\tau = 0.05$	38.3	0.78	2
$\tau = 0.15$	33.8	0.90	0
Primal update threshold ($\eta = 1.0$)			
$\eta = 0.0$	46.2	1.23	0
$\eta = 0.5$	43.6	1.16	0
$\eta = 0.9$	45.2	1.20	0
Number of sweeps ($\kappa = 1$, $\gamma = 0.1$)			
$\kappa = 5$, $\gamma = 0.1$	16.0	0.54	1
$\kappa = 5$, $\gamma = 0.2$	30.8	0.64	1
$\kappa = 5$, $\gamma = 0.5$	32.6	0.86	0
$\kappa = 5$, $\gamma = 1.0$	35.8	0.93	1
$\kappa = 5$, $\gamma = 1.5$	43.8	1.18	2

Table 7: Sensitivity with respect to parameters ω, ρ, α, γ, β, τ and η (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 $\kappa = 2$, in the best case with $\gamma = 0.2$, the parallel time was 9 % less than in the *Jacobi* base case. For other values of γ the performance slightly deteriorated as compared with the base case. For $\kappa = 5$, 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 κ 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 κ allows an increase in stepsize γ 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 ϕ 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

Problem	Rows	Karmarkar Speedup	Jacobi Speedup $\phi = 0.01$	$\phi = 0.001$
sctap3	1481	4.2	790.3	164.4
ship12l	1152	7.6	345.6	44.1
ship12s	1152	7.3	468.4	40.0
sctap2	1091	4.0	123.8	19.2
scfxm3	991	3.3	88.7	13.9
czprob	930	10.7	84.7	66.7
25fv47	822	6.9	1125.3	235.2
Average		6.3	355.7	82.3

Table 8: Speed-up factors for Karmarkar's algorithm and *Jacobi* base case relative to Minos 4.0; ϕ = 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

[1] I. Adler, M. Resende, G. Veiga and N. Karmarkar, An Implementation of the Karmarkar Algorithm for Linear Programming, *Mathematical Programming* 44 (1989) 297-335.

[2] D.P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods* (Academic Press, 1982).

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation* (Prentice-Hall, Englewood Cliffs, 1989).

[4] G Cohen and D.L. Zhu, "Decomposition-coordination methods in large scale optimization problems: the nondifferentiable case and the use of augmented Lagrangians," in: *Advances in Large Scale Systems, vol. 1*, J. B. Cruz (ed.), JAI Press 1984, pp. 203-266.

[5] G.B. Dantzig, *Linear Programming and Extensions* (Princeton University Press, Princeton, 1963).

[6] G.B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations Research* 8(1960) 101–111.

[7] Fiacco, A. and G. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* (John Wiley & Sons, 1968).

[8] M. Fortin and R. Glowinski, "On decomposition-coordination methods using an augmented Lagrangian," in: *Augmented Lagrangian Methods: Applications to the Numerical Solution of Bocudery-Value Problems*, M. Fortin and R. Glowinski (eds.), North-Holland, Amsterdam, 1983, pp. 97-146.

[9] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite-element approximations," *Comput. and Math. Appl.* 2(1976), pp. 17-40.

[10] R. Glowinski and A. Marocco, "Sur l'approximation par éléments finis d'ordre un et la résolution par pénalisation dualité d'une classe de problèmes de Dirichlet non linéaires," *Revue Française d'Automatique Informatique Recherche Opérationelle, Analyse Numérique*, R-2(1975), pp. 41-76.

[11] O. Güler, "Augmented Lagrangian algorithms for linear programming", *Journal of Optimization Theory and Applications* 75(1992) 445-470.

[12] M. R. Hestenes, "Multiplier and gradient methods", *Journal of Optimization Theory and Applications* 4(1969) 303-320.

[13] M. Kallio, D. Dykstra and C. Binkley (eds.), *The Global Forest Sector: An Analytival Perspective* (John Wiley & Sons, 1987).

[14] M. Kallio and S. Salo, "Tatonnement procedures for linearly constrained convex optimization," Helsinki School of Economics, 1992 (accepted for publication in *Management Science*).

[15] N. Karmarkar, "A New Polynomial Time Algorithm for Linear Programming," *Combinatorica* 4(1984) 373-395.

[16] J.M. Mulvey and A. Ruszczyński, "A diagonal quadratic approximation method for large scale linear programs," *Operations Research Letters* 12(1992) 205-315.

[17] J.M. Mulvey and A. Ruszczyński, "A new scenario decomposition method for large-scale stochastic optimization," technical report SOR 91-19, Department of Civil Engineering and Operations Research, Princeton University, Princeton 1991 (accepted for publication in *Operations Research*).

[18] B. Murtagh and M. Saunders, MINOS 5.1 User's Guide, Technical Report SOL 83-20R, Systems Optimization Labooratory, Stanford University, 1987.

[19] Netlib, LP Test Problems, Bell Laboratories.

[20] A.R. De Pierro and A.N. Iusem, "On the convergence of SOR- and JOR-type methods for convex linear complementarity problems", *Linear Algebra and Its Applications* 154-156(1991) 601-614.

[21] B.T. Polyak and N.V. Tretyakov, "An iterative method for linear programming and its economic interpretation", *Matecon* 10(1974) 81-100.

[22] M.J.D. Powell, "A method for nonlinear constraints in minimization problems", in: *Optimization*, R. Fletcher (ed.), Academic Press, New York 1969, pp. 283-298.

[23] R.T. Rockafellar, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming", *Mathematics of Operations Research* 1(1976) 97-116.

[24] A. Ruszczyński, "Augmented Lagrangian decomposition for sparse convex optimization", working paper WP 92-75, International Institute for Applied Systems Analysis, Laxenburg 1992 (accepted for publication in *Mathematics of Operations Research*).

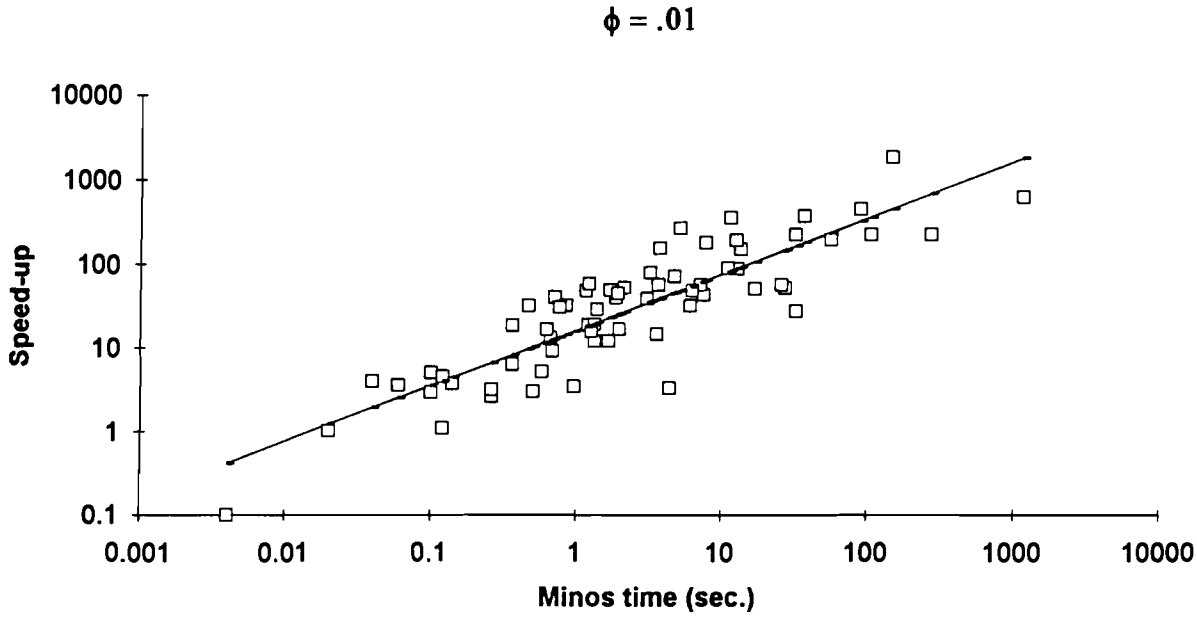
[25] P. Samuelson, "Spatial Price Equilibrium and Linear Programming", *The American Economic Review* 42(1952) 283-303.

[26] G. Stephanopoulos and W. Westerberg, "The use of Hestenes' method of multipliers to resolve dual gaps in engineering system optimization", *Journal of Optimization Theory and Applications*, 15(1975) 285-309.

[27] P. Tatjewski, "New dual-type decomposition algorithm for nonconvex separable optimization problems", *Automatica*, 25(1989) 233-242.

[28] N. Watanabe, Y. Nishimura and M. Matsubara, "Decomposition in large system optimization using the method of multipliers," *Journal of Optimization Theory and Applications*, 25(1978) 181-193.

Figure 1. Speed-up of *Jacobi* base case against Minos 5.1 as a function of serial run time for Minos; ϕ = relative tolerance.



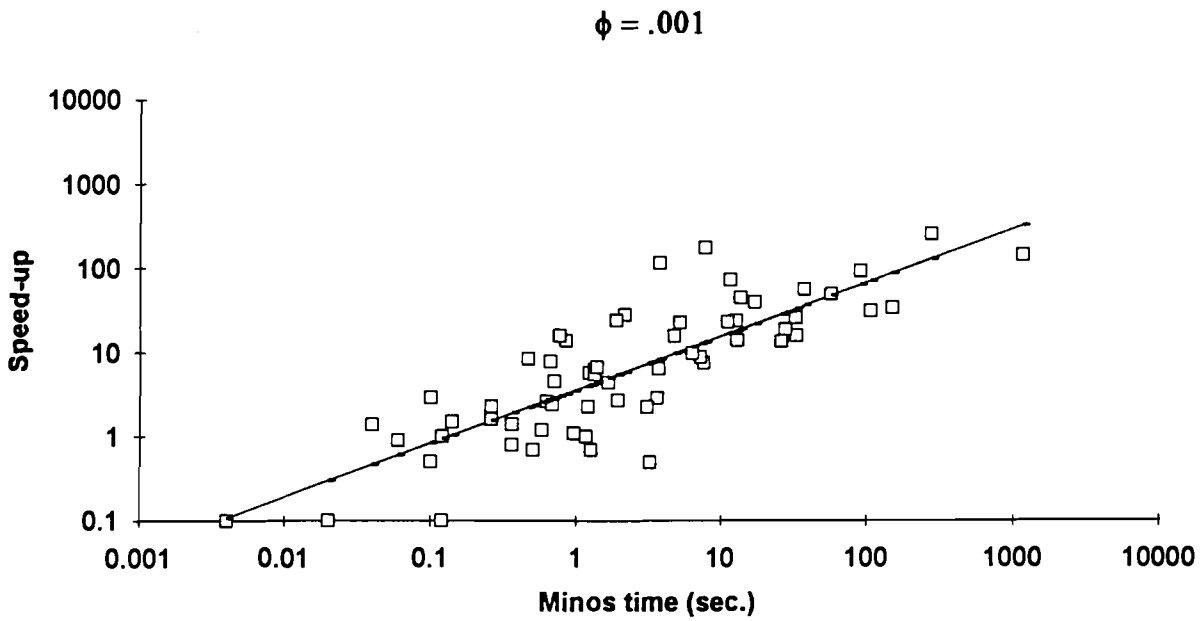


Figure 2. Speed-up *Jacobi* base case against Minos 5.1 as a function of the inverse of problem density; ϕ = relative tolerance.

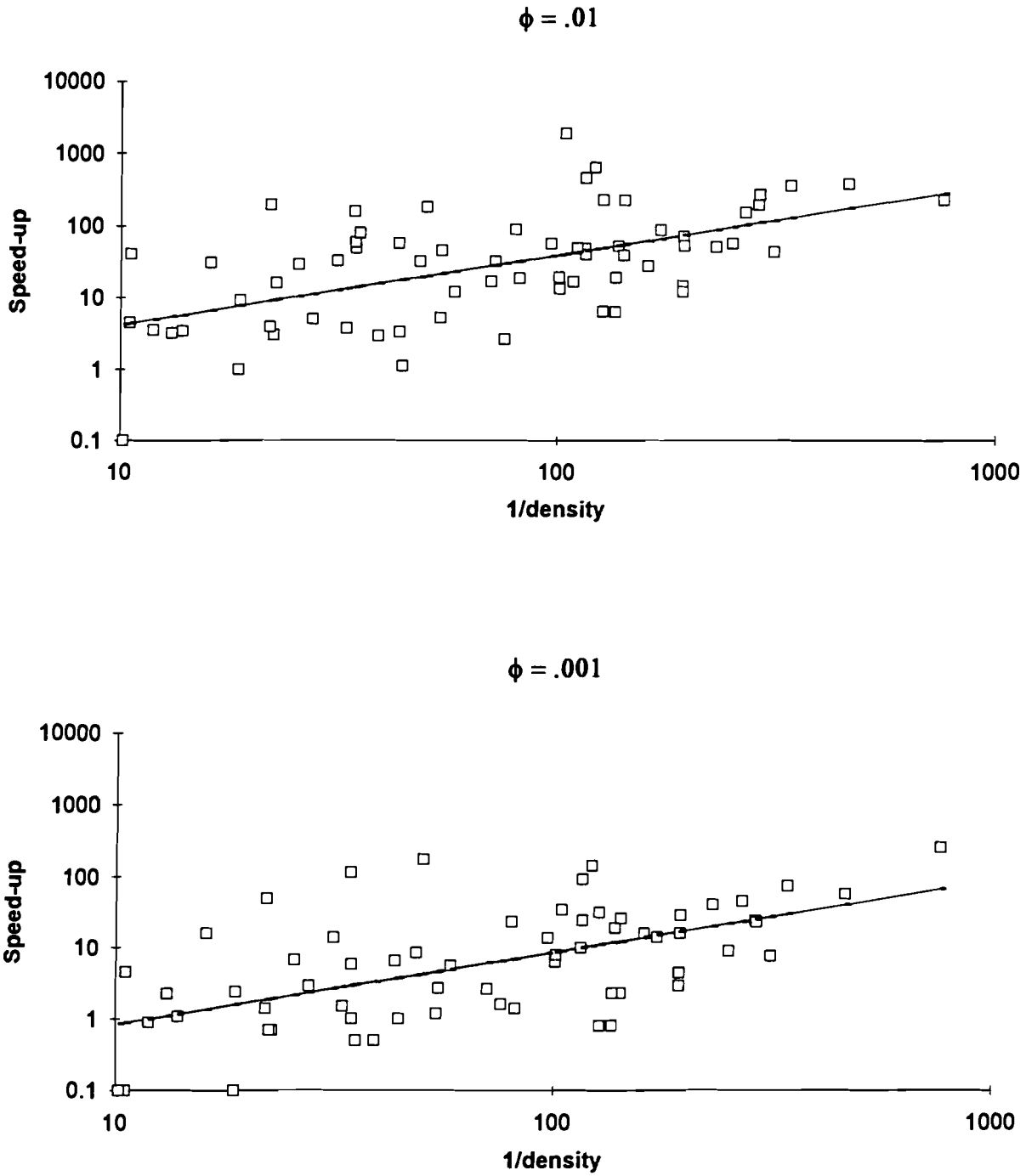
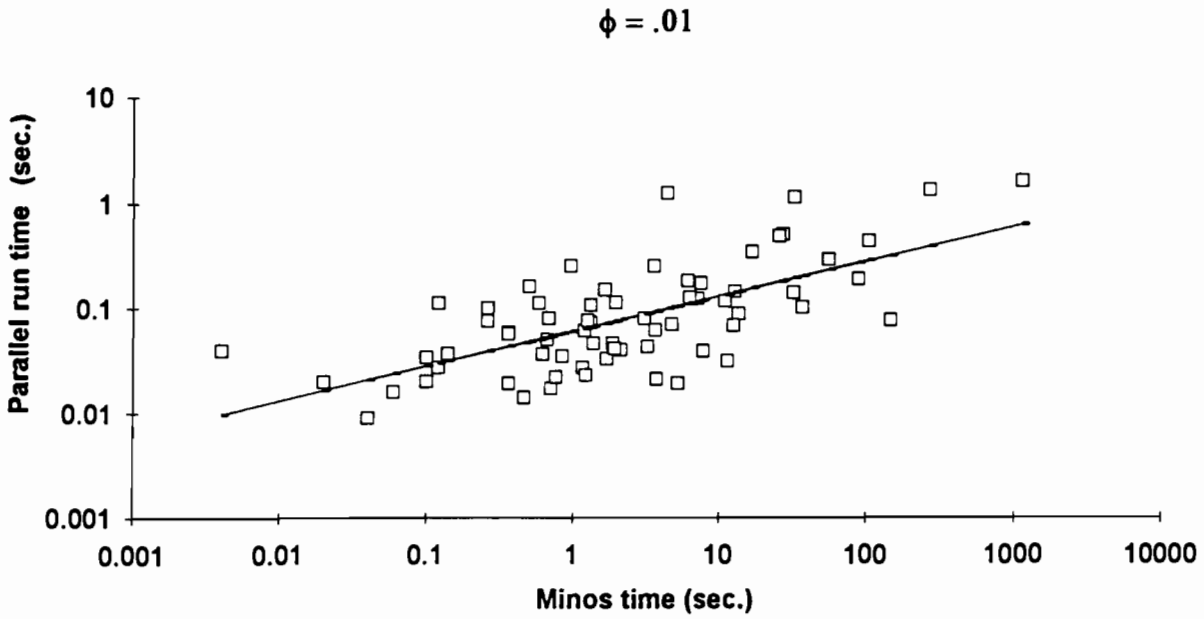


Figure 3. Parallel run time (sec) for *Jacobi* base case; ϕ = relative tolerance.



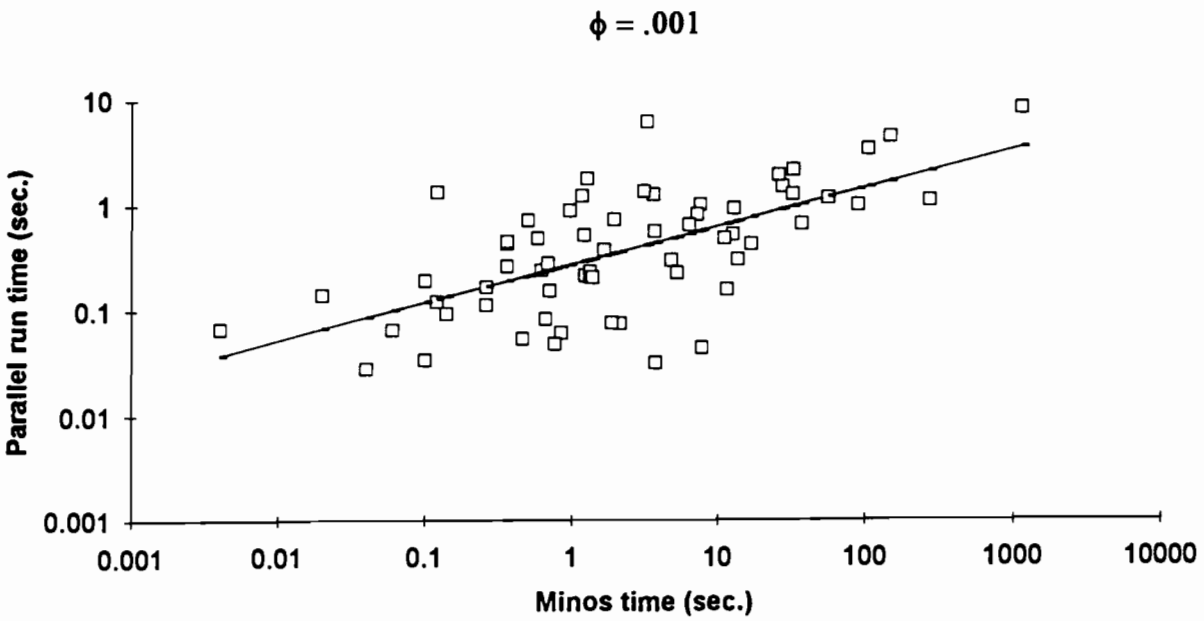


Figure 4. Number of iterations for *Jacobi* base case; ϕ = relative tolerance.

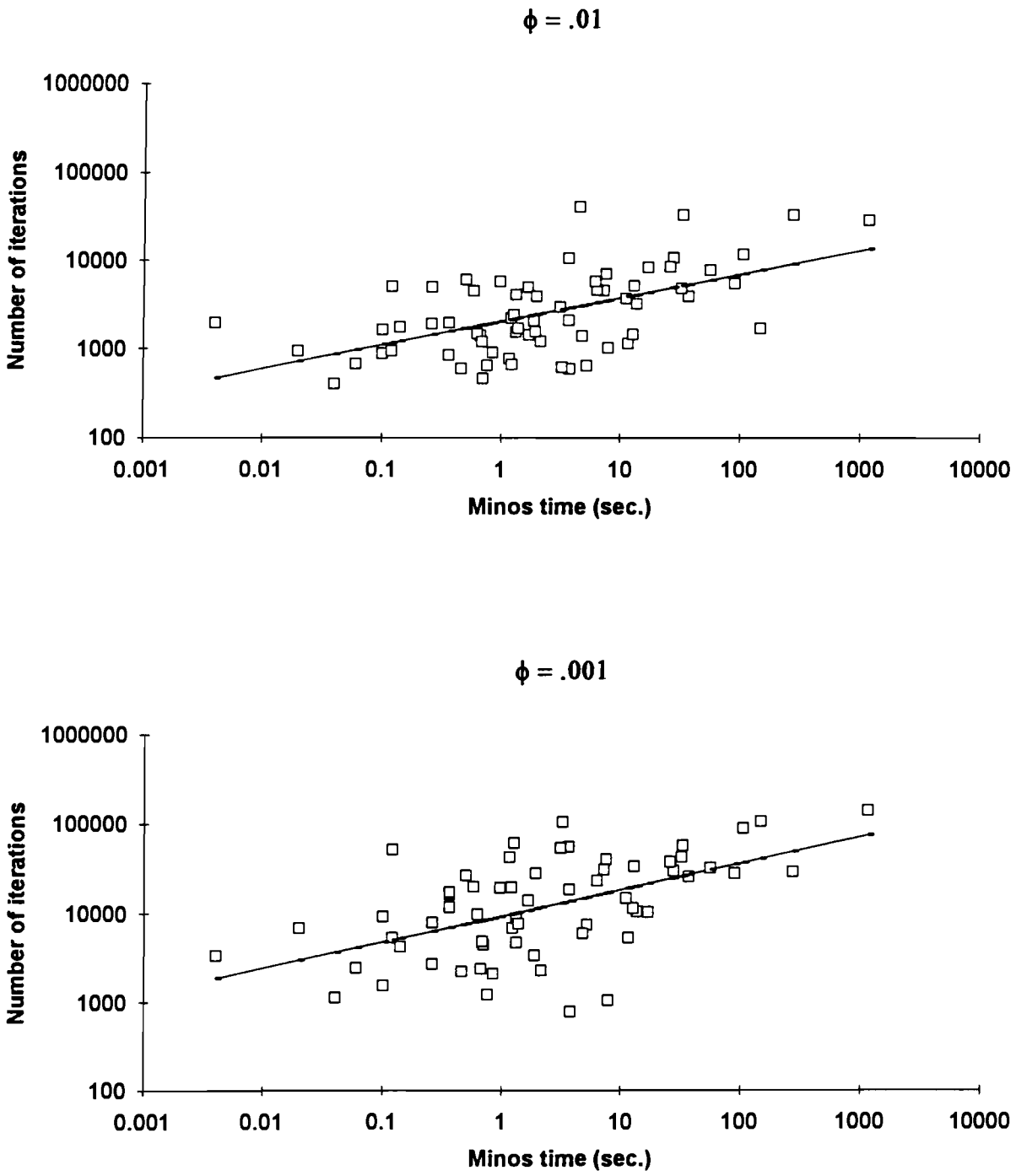
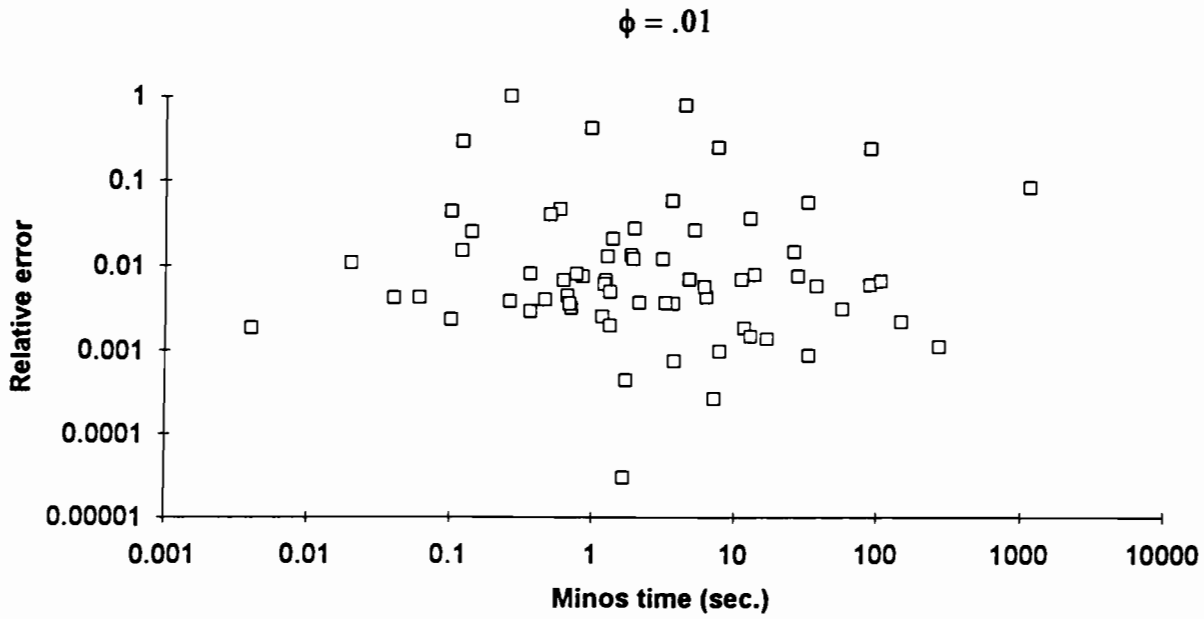


Figure 5. Relative error in the objective function value for *Jacobi* base case; ϕ = relative tolerance.



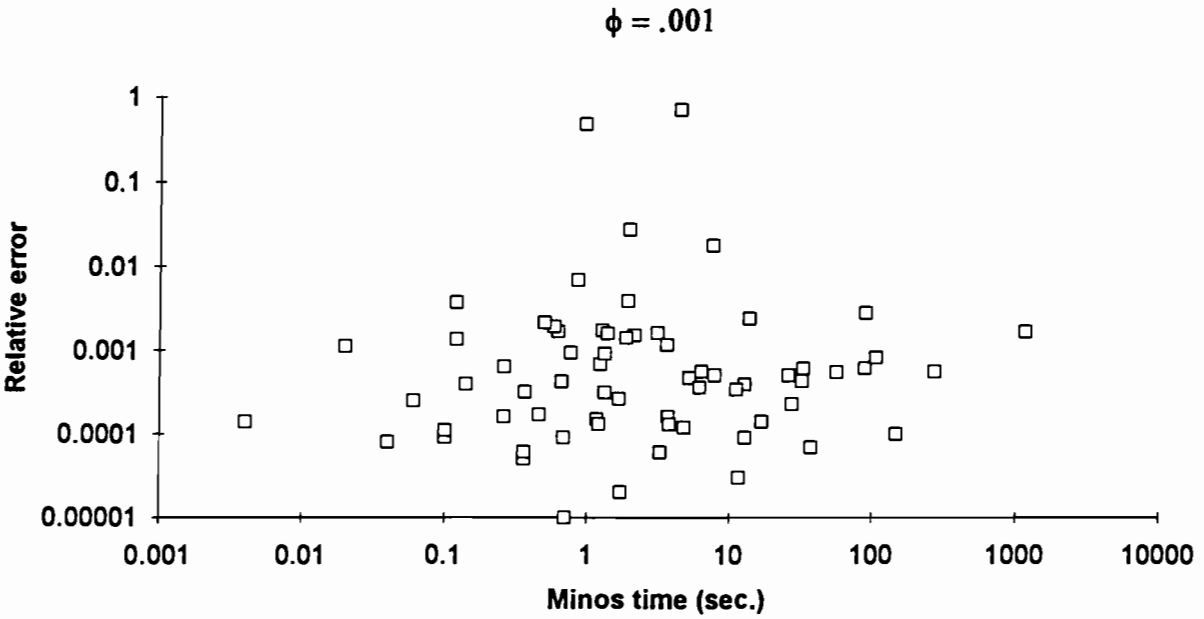


Figure 6. The run for problem *80bau3b*: Relative error of the objective function value as a function of iterations.

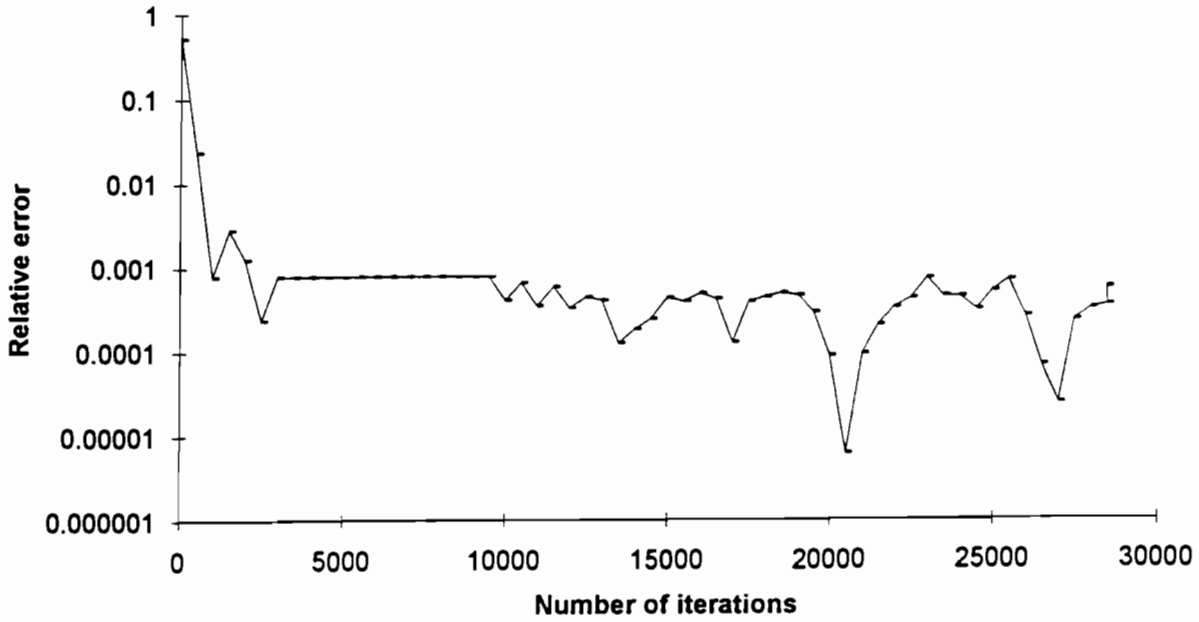


Figure 7. The run for problem *80bau3b*: Maximal relative primal and dual error as a function of iterations.

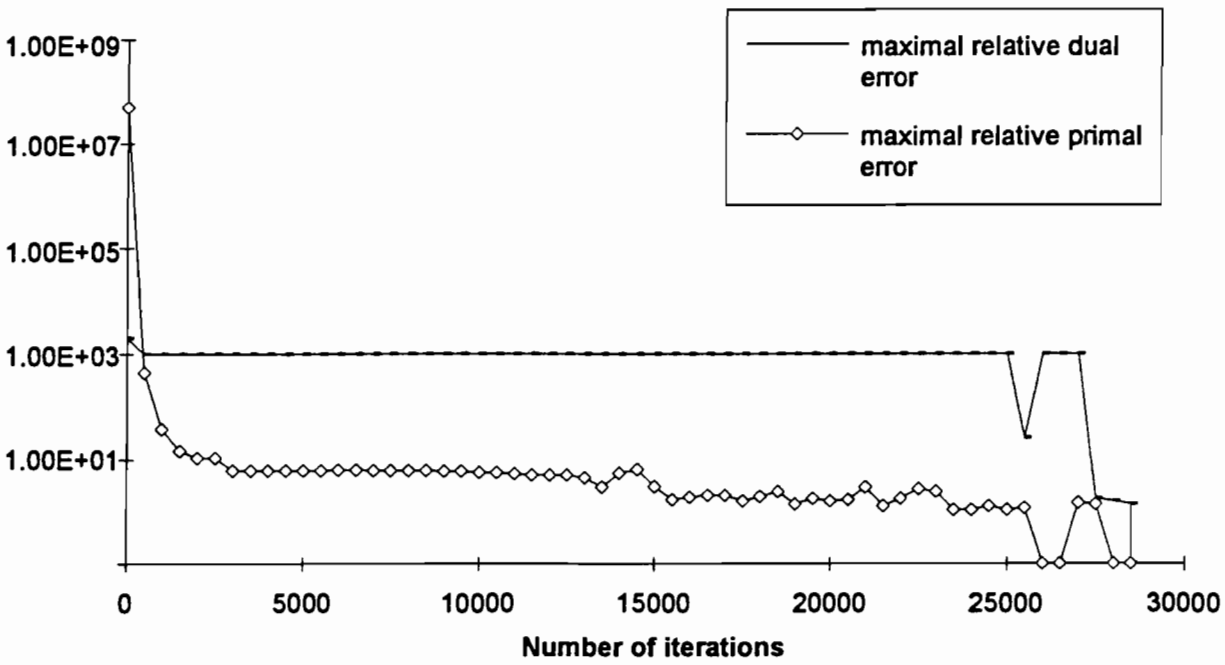


Figure 8. The run for problem *80bau3b*: The number of primal and dual infeasibilities as a function of iterations.

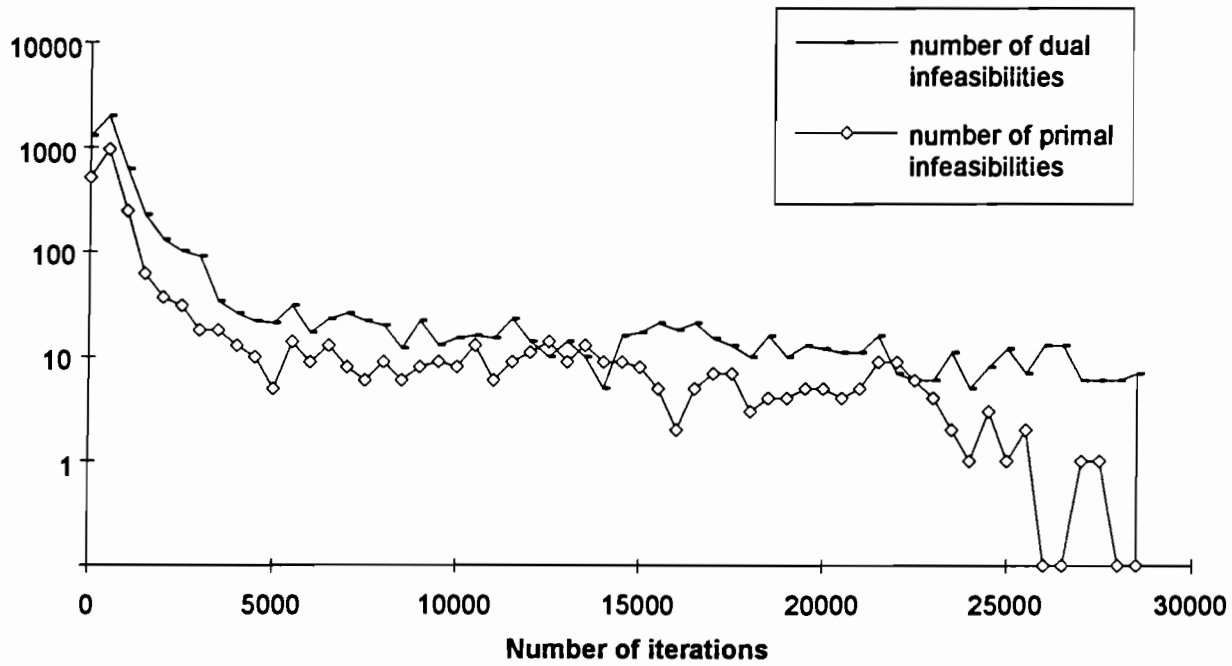


Figure 9. The run for problem *80bau3b*: Minimum, maximum and average value of the safe stepsize τ_j as a function of iterations.

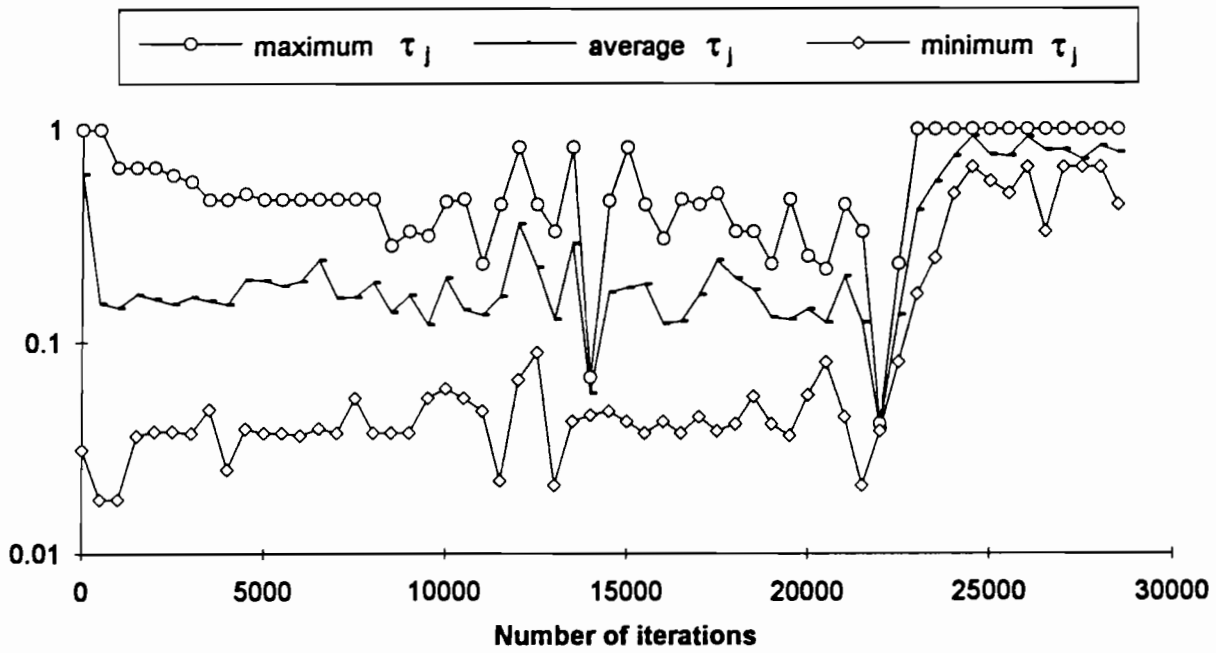


Figure 10. The run for problem *80bau3b*: The number of active rows and columns (in percent relative to m) as a function of iterations.

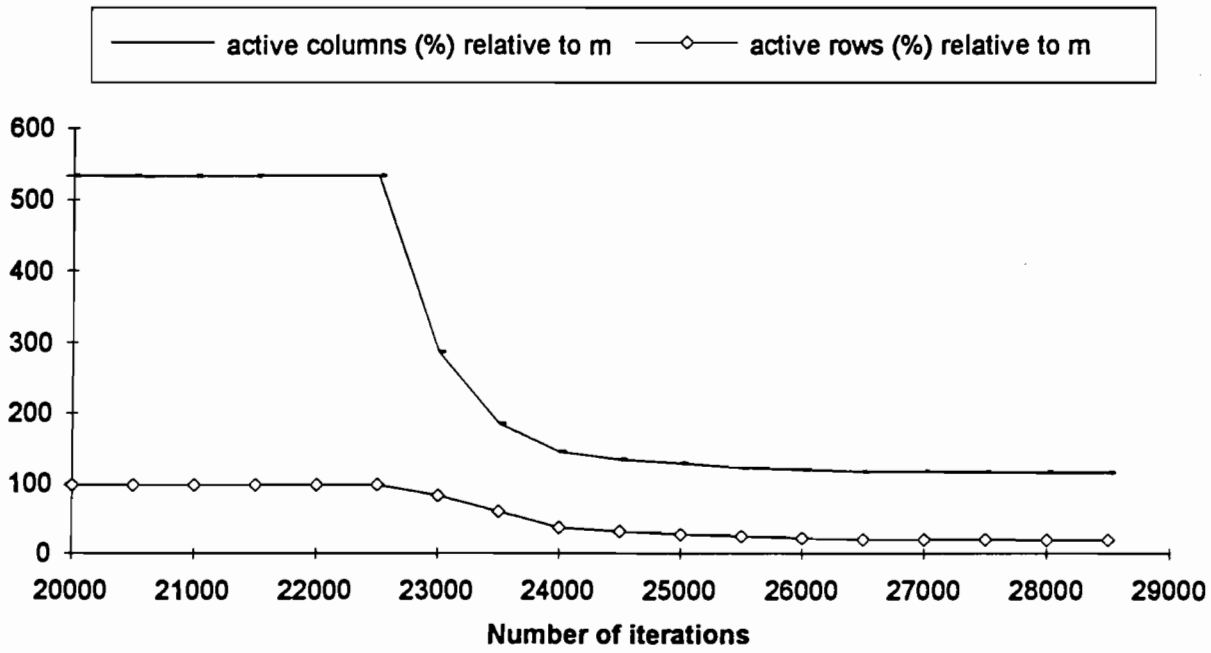


Figure 11. Speed-up of *Jacobi* base case against Gauss-Seidel in a serial computer; ϕ = relative tolerance.