Parallel Relaxation Method for Quadratic Programming Problems with Interval Constraints

Takanobu Sugimoto[1], Masao Fukushima[2] and Toshihide Ibaraki[3]

[1]Ministry of International Trade and Industry, Tokyo 100, Japan

[2]Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-01, Japan

[3]Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan

Abstract. Optimization problems with interval constraints are encountered in various fields such as network flows and computer tomography. As these problems are usually very large, they are not easy to solve without taking their sparsity into account. Recently "row-action methods", which originate from the classical Hildreth's method for quadratic programming problems, have drawn much attention, since they are particularly useful for large and sparse problems. Various row-action methods have already been proposed for optimization problems with interval constraints, but they mostly belong to the class of sequential methods based on the Gauss-Seidel and SOR methods.

In this paper, we propose a highly parallelizable method for solving those problems, which may be regarded as an application of the Jacobi method to the dual of the original problems. We prove convergence of the algorithm and report some computational results to demonstrate its effectiveness.

Key Words. Convex programming, interval constraints, Jacobi method, row-action method, parallel algorithm.

Abreviated Title. Parallel algorithm for interval constraints.

1 Introduction

Consider the strictly convex quadratic programming problem

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}(x-x^0)^T Q(x-x^0) \\ \text{subject to} \quad & \gamma \le Ax \le \delta, \end{aligned} \tag{1}$$

where $A = (a_{ij})$ is a given $m \times n$ matrix whose ith row is a_i^T, $x^0 \in R^n, \gamma, \delta \in R^m$ are given vectors, Q is a given $n \times n$ symmetric positive-definite matrix and the superscript T denotes transposition. We assume that matrix A does not contain any row of which elements are all zero. The pairs of inequality constraints in problem (1) are referred to as *interval constraints*. Interval constraints often appear in optimization problems that arise in various fields such as network flows and computer tomography.

Recently, various row-action methods [2, 3, 16], which originate from the classical Hildreth's method [7], have drawn much attention. Those methods are particularly useful for large and sparse problems, because they act upon rows of the original coefficient matrix one at a time. They are adaptations of coordinate descent methods such as Gauss-Seidel method or its variants, for solving the dual of a given quadratic programming problem. Since problem (1) may be written as

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}(x-x^0)^T Q(x-x^0) \\ \text{subject to} \quad & Ax \le \delta, \\ & -Ax \le -\gamma, \end{aligned} \tag{2}$$

ordinary row-action methods can be applied to this problem in a straightforward manner. In this case, however, we must treat $2m$ dual variables.

By taking into account the special structure of problem (1), Herman and Lent [6] extended Hildreth's algorithm to deal with interval constraints directly. The principal advantage of the latter method over the straightforward application of Hildreth's algorithm to the equivalent problem (2) lies in the fact that it needs only one dual variable, rather than two dual variables, for each pair of constraints, thereby economizing the number of dual variables by half (see also [3]). To accelerate convergence, Shimazu, Fukushima and Ibaraki [15] have recently proposed an SOR method that can deal with the interval constraints in a direct manner. Iusem and Zenios [10] also proposed an underrelaxation method for a class of interval constrained nonlinear programming problems.

Though the methods mentioned above can sometimes be implemented on a parallel computer by taking advantage of a certain special structure of the problem to be solved, they are essentially sequential algorithms of Gauss-Seidel or SOR type. In this paper, we propose a highly parallelizable method for solving the interval constrained problem (1), which is an adaptation of Jacobi method for solving the dual. Since each dual variable can be updated independently at each iteration, this method can be efficiently executed on a parallel computer by assigning them to different processors. It should be mentioned that a Jacobi-type method has recently been proposed by Iusem and De Pierro [9] to solve ordinary (non-interval) inequality constrained quadratic programming problems (see also [5]). So one might think that the method presented here could be obtained by applying the method of Iusem and De Pierro to problem (2) in a straightforward manner. But, as shown later (cf. Algorithm 3.1), the proposed algorithm can be regarded as a composition of parallel and sequential procedures, so it has a quite different nature from the method of Iusem and De Pierro.

This paper is organized as follows. In Section 2, a parallel relaxation method for problem (1) is presented. In Section 3, the proposed algorithm is shown to converge to the solution of (1). In Section 4, computational results with the proposed algorithm are reported. Finally Section 5 concludes the paper.

2 Parallel Relaxation Method

In this section, we state an algorithm for solving the interval constrained problem (1).

Algorithm 2.1.

Initialization: Let $(x^{(0)}, u^{(0)}) := (x^0, 0)$, $k := 0$ and choose a relaxation parameter $\omega > 0$.

Iteration k: (i) For $i = 1, \cdots, m$, let

$$u_i^{(k+1)} := u_i^{(k)} - c_i^{(k)},$$

with

$$c_i^{(k)} := \mathrm{mid}\left\{u_i^{(k)}, \omega\Delta_i^{(k)}, \omega\Gamma_i^{(k)}\right\},$$

where mid denotes the median of three numbers and

$$\Delta_i^{(k)} := \frac{\delta_i - a_i^T x^{(k)}}{\alpha_i},$$

$$\Gamma_i^{(k)} := \frac{\gamma_i - a_i^T x^{(k)}}{\alpha_i}.$$

(ii) Let

$$x^{(k+1)} := x^{(k)} + Q^{-1} \sum_{i=1}^{m} c_i^{(k)} a_i,$$

where

$$\alpha_i = a_i^T Q^{-1} a_i, \qquad i = 1, \cdots, m. \tag{3}$$

Note that α_i are all positive, since Q is positive definite and $a_i \neq 0$ by assumption. Note also that, since $\gamma_i \leq \delta_i$, the following inequalities are always satisfied:

$$\Gamma_i^{(k)} \leq \Delta_i^{(k)}, \qquad i = 1, \cdots, m. \tag{4}$$

3 Convergence Theorems

In this section, we prove that Algorithm 2.1 described in the previous section converges to the solution of problem (1). To this end, it will be helpful to consider the dual of problem (1)

$$\begin{aligned} &\text{minimize} \quad \phi(z) \\ &\text{subject to} \quad z \geq 0, \end{aligned} \tag{5}$$

where $\phi : R^{2m} \rightarrow R$ is a convex quadratic function defined by

$$\phi(z) = \frac{1}{2} z^T \hat{A} Q^{-1} \hat{A}^T z + z^T (b - \hat{A} x^0), \tag{6}$$

$\hat{A}$ is the $2m \times n$ matrix

$$\hat{A} = (a_1, -a_1, a_2, -a_2, \cdots, a_m, -a_m)^T, \tag{7}$$

b is the $2m$-vector

$$b = (\delta_1, -\gamma_1, \delta_2, -\gamma_2, \cdots, \delta_m, -\gamma_m)^T, \tag{8}$$

and z is the $2m$-vector

$$z = (z_1^+, z_1^-, z_2^+, z_2^-, \cdots, z_m^+, z_m^-)^T. \tag{9}$$

Note that (z_i^+, z_i^-) is a pair of dual variables associated with the ith pair of the interval constraints of (1), i.e., z_i^+ and z_i^- correspond to the constraints $a_i^T x \leq \delta_i$ and $-a_i^T x \leq -\gamma_i$, respectively. The proof of convergence consists of the following steps. First, we present an equivalent algorithm to Algorithm 2.1, which generates sequences $\{x^{(k)}\}$ and $\{z^{(k)}\}$ such that $z^{(k)} \geq 0$ for all k. This reduction is similar to the one by Censor and Lent [3] (see also [15]). Next, we prove convergence of the equivalent algorithm, thereby establishing convergence of Algorithm 2.1.

3.1 Equivalent Algorithm to Algorithm 2.1

Now let us consider the following algorithm.

Algorithm 3.1.

Initialization: Let $(x^{(0)}, z^{(0)}) := (x^0, 0)$, $k := 0$ and choose a relaxation parameter $\omega > 0$.

Iteration k: (i) For $i = 1, \cdots, m$,

$$
\begin{aligned}
&\text{if } z_i^{+(k)} \geq z_i^{-(k)} \text{ then} \\
&\qquad c_i^{+(k)} := \min\left\{z_i^{+(k)}, \omega\Delta_i^{(k)}\right\}, \\
&\qquad c_i^{-(k)} := \min\left\{z_i^{-(k)}, -\omega\Gamma_i^{(k)} + c_i^{+(k)}\right\} \\
&\text{else} \\
&\qquad c_i^{-(k)} := \min\left\{z_i^{-(k)}, -\omega\Gamma_i^{(k)}\right\}, \\
&\qquad c_i^{+(k)} := \min\left\{z_i^{+(k)}, \omega\Delta_i^{(k)} + c_i^{-(k)}\right\} \\
&\text{endif}
\end{aligned}
$$

$$
\begin{aligned}
z_i^{+(k+1)} &:= z_i^{+(k)} - c_i^{+(k)}, \\
z_i^{-(k+1)} &:= z_i^{-(k)} - c_i^{-(k)},
\end{aligned}
$$

where

$$
\begin{aligned}
\Delta_i^{(k)} &:= \frac{\delta_i - a_i^T x^{(k)}}{\alpha_i}, \\
\Gamma_i^{(k)} &:= \frac{\gamma_i - a_i^T x^{(k)}}{\alpha_i}.
\end{aligned}
$$

(ii) Let

$$
x^{(k+1)} := x^{(k)} + Q^{-1}\sum_{i=1}^{m}\left(c_i^{+(k)} - c_i^{-(k)}\right) a_i.
$$

Lemma 3.1. *Let $\{x^{(k)}\}$ and $\{z^{(k)}\}$ be generated by Algorithm 3.1. Then for all k, we have*

$$x^{(k)} = x^0 - Q^{-1}\hat{A}^T z^{(k)}, \tag{10}$$

$$z^{(k)} \geq 0, \tag{11}$$

$$z_i^{+(k)} \cdot z_i^{-(k)} = 0, \quad i = 1, \cdots, m. \tag{12}$$

Proof. First notice that (10) and (11) directly follow from the manner in which $\{x^{(k)}\}$ and $\{z^{(k)}\}$ are updated in the algorithm. We prove (12) by induction. For $k = 0$, it trivially holds. For each i, we assume $z_i^{+(k)} \cdot z_i^{-(k)} = 0$ and show that it is also true for $k+1$. Without loss of generality, we may only consider the case where $z_i^{+(k)} \geq z_i^{-(k)}$, because a parallel argument is valid for the opposite case. Note that, when $z_i^{+(k)} \geq z_i^{-(k)}$, (11) implies $z_i^{+(k)} \geq 0$ and $z_i^{-(k)} = 0$. Moreover, if $z_i^{+(k)} \geq \omega\Delta_i^{(k)}$ holds, then we have $c_i^{+(k)} = \omega\Delta_i^{(k)}$ and $c_i^{-(k)} = \min\left\{0, \omega(\Delta_i^{(k)} - \Gamma_i^{(k)})\right\} = 0$, where the last equality follows from (4). Therefore we must have $z_i^{-(k+1)} = 0$. On the other hand, if $z_i^{+(k)} < \omega\Delta_i^{(k)}$ holds, then we have $c_i^{+(k)} = z_i^{+(k)}$, which in turn implies $z_i^{+(k+1)} = 0$. Thus (12) is satisfied for $k+1$. $\square$

For each i, either $z_i^{+(k)} = 0$ or $z_i^{-(k)} = 0$ must always hold by (12). Moreover, we can deduce the following relations:

If $z_i^{+(k)} \geq z_i^{-(k)}$, i.e., $z_i^{+(k)} \geq 0, z_i^{-(k)} = 0$, then

$$(c_i^{+(k)}, c_i^{-(k)}) = \begin{cases} (\omega\Delta_i^{(k)}, 0), & \text{if } z_i^{+(k)} \geq \omega\Delta_i^{(k)}, \\ (z_i^{+(k)}, 0), & \text{if } \omega\Delta_i^{(k)} \geq z_i^{+(k)} \geq \omega\Gamma_i^{(k)}, \\ (z_i^{+(k)}, -\omega\Gamma_i^{(k)} + z_i^{+(k)}), & \text{if } \omega\Gamma_i^{(k)} \geq z_i^{+(k)}. \end{cases} \tag{13}$$

If $z_i^{+(k)} \leq z_i^{-(k)}$, i.e., $z_i^{+(k)} = 0, z_i^{-(k)} \geq 0$, then

$$(c_i^{+(k)}, c_i^{-(k)}) = \begin{cases} (0, -\omega\Gamma_i^{(k)}), & \text{if } z_i^{-(k)} \geq -\omega\Gamma_i^{(k)}, \\ (0, z_i^{-(k)}), & \text{if } -\omega\Gamma_i^{(k)} \geq z_i^{-(k)} \geq -\omega\Delta_i^{(k)}, \\ (\omega\Delta_i^{(k)} + z_i^{-(k)}, z_i^{-(k)}), & \text{if } -\omega\Delta_i^{(k)} \geq z_i^{-(k)}. \end{cases} \tag{14}$$

The next theorem demonstrates the equivalence between Algorithm 2.1 and Algorithm 3.1.

Theorem 3.1. *Algorithm 2.1 and Algorithm 3.1 are equivalent in the sense that*

$$u_i^{(k)} = z_i^{+(k)} - z_i^{-(k)}, \quad i = 1, 2, \cdots, m \tag{15}$$

$$c_i^{(k)} = c_i^{+(k)} - c_i^{-(k)}, \quad i = 1, 2, \cdots, m \tag{16}$$

hold for all k and the two algorithms generate an identical sequence $\{x^{(k)}\}$.

Proof. The proof is by induction. Since $u^{(0)} = 0$ and $z^{(0)} = 0$, (15) is obvious for $k = 0$. Now assuming that (15) is true for k, we shall show that (16) and (15) hold for k and $k+1$, respectively.

Since either $z_i^{+(k)} = 0$ or $z_i^{-(k)} = 0$ holds, (13) and (14) imply

$$c_i^{+(k)} - c_i^{-(k)} = \text{mid}\left\{z_i^{+(k)} - z_i^{-(k)}, \omega\Delta_i^{(k)}, \omega\Gamma_i^{(k)}\right\}. \tag{17}$$

Moreover from the manner in which $z_i^{+(k)}$ and $z_i^{-(k)}$ are updated, it follows that

$$z_i^{+(k+1)} - z_i^{-(k+1)} = z_i^{+(k)} - z_i^{-(k)} - \left(c_i^{+(k)} - c_i^{-(k)}\right). \tag{18}$$

On the other hand, we have

$$c_i^{(k)} = \text{mid}\left\{u_i^{(k)}, \omega\Delta_i^{(k)}, \omega\Gamma_i^{(k)}\right\} \tag{19}$$

and

$$u_i^{(k+1)} = u_i^{(k)} - c_i^{(k)}. \tag{20}$$

Thus, since (15) is true for k, (17) and (19) imply that (16) holds for k. Moreover, (18) and (20) together with (16) and (15) imply that (15) is true for $k+1$. □

3.2 Convergence of Algorithm 3.1

In this subsection, we prove convergence of Algorithm 3.1. First, we show that this algorithm can be regarded as a matrix splitting method for solving the linear complementarity problem equivalent to the dual problem (5). We then examine conditions under which this algorithm is convergent.

Observe that problem (5) is equivalent to the linear complementarity problem (LCP)

$$Mz + q \geq 0, \quad z \geq 0, \quad z^T(Mz + q) = 0, \tag{21}$$

where

$$M = \hat{A}Q^{-1}\hat{A}^T, \tag{22}$$

$$q = b - \hat{A}x^0. \tag{23}$$

Note that the positive definiteness of Q implies that the matrix M defined by (22) is positive semidefinite. Note also that the function ϕ defined by (6) can be written as

$$\phi(z) = \frac{1}{2}z^T M z + q^T z. \tag{24}$$

A general matrix splitting method for solving the LCP (21) may be stated as follows:

Algorithm 3.2.

Initialization: Let $z^{(0)} := 0$ and $k := 0$.

Iteration k: Split the matrix M as the sum

$$M = B^{(k)} + C^{(k)} \tag{25}$$

and solve the LCP

$$B^{(k)}z + q^{(k)} \geq 0, \quad z \geq 0, \quad z^T(B^{(k)}z + q^{(k)}) = 0, \tag{26}$$

where $q^{(k)} = C^{(k)}z^{(k)} + q$. Let $z^{(k+1)}$ be a solution of (26).

Note that, unlike ordinary matrix splitting methods (see, for example, [4, Chapter 5]), Algorithm 3.2 allows the splitting (25) to vary with iteration k.

The next theorem shows that Algorithm 3.1 may be regarded as a matrix splitting method (Algorithm 3.2) by suitable choice of $B^{(k)}$.

Theorem 3.2. *Algorithm 3.2 is reduced to Algorithm 3.1 if we choose for each k*

$$B^{(k)} = \omega^{-1}(D + K^{(k)}), \tag{27}$$

where D and $K^{(k)}$ are $2m \times 2m$ block diagonal matrices such that

$$D = \begin{pmatrix} D_1 & & 0 \\ & \ddots & \\ 0 & & D_m \end{pmatrix}, \quad K^{(k)} = \begin{pmatrix} K_1^{(k)} & & 0 \\ & \ddots & \\ 0 & & K_m^{(k)} \end{pmatrix}, \tag{28}$$

with

$$D_i = \begin{pmatrix} \alpha_i & 0 \\ 0 & \alpha_i \end{pmatrix}, \quad K_i^{(k)} = \begin{cases} \begin{pmatrix} 0 & 0 \\ -\alpha_i & 0 \end{pmatrix}, & \text{if } z_i^{+(k)} \geq z_i^{-(k)}, \\ \\ \begin{pmatrix} 0 & -\alpha_i \\ 0 & 0 \end{pmatrix}, & \text{otherwise,} \end{cases} \tag{29}$$

and $\alpha_i = a_i^T Q^{-1} a_i$ *for all* $i = 1, \cdots, m$ *(cf. (3)).*

Proof. Let $B^{(k)}$ be given by (27). Then, since D and $K^{(k)}$ are block diagonal, we may write, after some calculations, the solution $z^{(k+1)} = (z_1^{+(k+1)}, z_1^{-(k+1)}, \cdots, z_m^{+(k+1)}, z_m^{-(k+1)})$ of the subproblem (26) in Algorithm 3.2 explicitly as follows: For each $i = 1, \cdots, m$, if $z_i^{+(k)} \geq z_i^{-(k)}$ then

$$z_i^{+(k+1)} = \left(z_i^{+(k)} - \frac{\omega}{\alpha_i} \left(a_i^T Q^{-1} \hat{A}^T z^{(k)} + \delta_i - a_i^T x^0 \right) \right)_+, \tag{30}$$

$$z_i^{-(k+1)} = \left(z_i^{-(k)} - \frac{\omega}{\alpha_i} \left(-a_i^T Q^{-1} \hat{A}^T z^{(k)} - \gamma_i + a_i^T x^0 - \frac{\alpha_i}{\omega} \left(z_i^{+(k+1)} - z_i^{+(k)} \right) \right) \right)_+ \tag{31}$$

otherwise

$$z_i^{-(k+1)} = \left(z_i^{-(k)} - \frac{\omega}{\alpha_i} \left(-a_i^T Q^{-1} \hat{A}^T z^{(k)} - \gamma_i + a_i^T x^0 \right) \right)_+,$$

$$z_i^{+(k+1)} = \left(z_i^{+(k)} - \frac{\omega}{\alpha_i} \left(a_i^T Q^{-1} \hat{A}^T z^{(k)} + \delta_i - a_i^T x^0 - \frac{\alpha_i}{\omega} \left(z_i^{-(k+1)} - z_i^{-(k)} \right) \right) \right)_+,$$

where $(\cdot)_+ = \max\{0, \cdot\}$.

In what follows, we shall only consider the case where $z_i^{+(k)} \geq z_i^{-(k)}$. The opposite case can be treated in an analogous manner. Now let

$$x^{(k)} = x^0 - Q^{-1} \hat{A}^T z^{(k)}, \tag{32}$$

$$\Delta_i^{(k)} = \frac{\delta_i - a_i^T x^{(k)}}{\alpha_i}, \tag{33}$$

$$\Gamma_i^{(k)} = \frac{\gamma_i - a_i^T x^{(k)}}{\alpha_i}. \tag{34}$$

Then, using (32)–(34), we may rewrite (30) and (31) as follows: For $i = 1, \cdots, m$,

$$\begin{aligned}
z_i^{+(k+1)} &= \left(z_i^{+(k)} - \frac{\omega}{\alpha_i} \left(\delta_i - a_i^T x^{(k)} \right) \right)_+ \\
&= \max \left\{ 0, z_i^{+(k)} - \omega \Delta_i^{(k)} \right\} \\
&= z_i^{+(k)} - \min \left\{ z_i^{+(k)}, \omega \Delta_i^{(k)} \right\}, \\
z_i^{-(k+1)} &= \left(z_i^{-(k)} - \frac{\omega}{\alpha_i} \left(-\gamma_i + a_i^T x^{(k)} - \frac{\alpha_i}{\omega} \left(z_i^{+(k+1)} - z_i^{+(k)} \right) \right) \right)_+ \\
&= \max \left\{ 0, z_i^{-(k)} + \omega \Gamma_i^{(k)} + \left(z_i^{+(k+1)} - z_i^{+(k)} \right) \right\} \\
&= z_i^{-(k)} - \min \left\{ z_i^{-(k)}, -\omega \Gamma_i^{(k)} - \left(z_i^{+(k+1)} - z_i^{+(k)} \right) \right\},
\end{aligned}$$

which may further be written

$$z_i^{+(k+1)} = z_i^{+(k)} - c_i^{+(k)}, \tag{35}$$

$$z_i^{-(k+1)} = z_i^{-(k)} - c_i^{-(k)}, \tag{36}$$

where

$$\begin{aligned}
c_i^{+(k)} &:= \min \left\{ z_i^{+(k)}, \omega \Delta_i^{(k)} \right\}, \\
c_i^{-(k)} &:= \min \left\{ z_i^{-(k)}, -\omega \Gamma_i^{(k)} + c_i^{+(k)} \right\}.
\end{aligned}$$

Moreover, we have

$$\begin{aligned}
x^{(k+1)} &= x^0 - Q^{-1} \sum_{i=1}^{m} \left(z_i^{+(k+1)} - z_i^{-(k+1)} \right) a_i \\
&= x^0 - Q^{-1} \sum_{i=1}^{m} \left(z_i^{+(k)} - z_i^{-(k)} \right) a_i + Q^{-1} \sum_{j=1}^{m} \left(c_i^{+(k)} - c_i^{-(k)} \right) a_i \\
&= x^{(k)} + Q^{-1} \sum_{i=1}^{m} \left(c_i^{+(k)} - c_i^{-(k)} \right) a_i,
\end{aligned}$$

where the first and the third equalities follow from (32), while the second follows from (35) and (36). Thus we have shown that Algorithm 3.1 is derived from Algorithm 3.2 by setting $B^{(k)}$ as (27). $\square$

Note that the splitting $M = B^{(k)} + C^{(k)}$ determined by (27) depends on the current value of $z^{(k)}$. Specifically, the ith diagonal block of $B^{(k)}$ is lower triangular if $z_i^{+(k)} \geq z_i^{-(k)}$, and upper triangular otherwise. This matrix splitting method is different from

other methods with $M = B + C$ such as the block method ($B = \tilde{D}$) and the block SOR method ($B = \omega^{-1}\tilde{D} + \tilde{L}$), where $\tilde{D}$ and $\tilde{L}$ are the block diagonal and strictly block lower triangular parts of M, respectively (see [4, page 398]). (Note that the block diagonal part of M consists of the 2×2 matrices

$$\tilde{D}_i = \begin{pmatrix} \alpha_i & -\alpha_i \\ -\alpha_i & \alpha_i \end{pmatrix},$$

which differs from the block diagonal matrix $D + K^{(k)}$ that appears in the definition (27) of $B^{(k)}$.) The matrix splitting method with $M = B^{(k)} + C^{(k)}$ is closer in spirit to the parallel SOR method of Mangasarian and De Leone [12], which may be realized by the splitting $M = B + C$ with

$$B = \omega^{-1}D + L, \tag{37}$$

where D is the same as in (27) and L is the block diagonal matrix whose diagonal blocks consist of the strictly lower triangular part of the matrices $\tilde{D}_i$ defined above. So, L is very similar to the matrix $K^{(k)}$ given by (28) and (29). But, apart from the fact that the matrix B does not depend on k, it contains the relaxation parameter ω only in its diagonal entries (cf. (27) and (37)).

Let us now turn our attention to convergence conditions for Algorithm 3.2 with the splitting $M = B^{(k)} + C^{(k)}$ determined by (27) – (29). Recently a number of important results have been obtained regarding matrix splitting methods for the linear complementarity problem [8, 11] (see also [4]). In particular, it follows from Iusem [8, Theorem 3] that Algorithm 3.2 is convergent to a solution of the problem if the function ϕ defined by (24) is bounded below for $z \geq 0$ and if the splitting $M = B^{(k)} + C^{(k)}$ satisfies the following conditions:

(a) LCP (26) has a solution for each k;

(b) $\{C^{(k)}\}$ is bounded;

(c) $\{B^{(k)} - C^{(k)}\}$ is uniformly positive definite.

First notice that, if the original problem (1) is feasible, then it has a unique optimal solution and its dual problem (5) also has an optimal solution. In this case, the function

ϕ is clearly bounded below on $z \geq 0$. Moreover, it is not difficult to see that conditions (a) and (b) hold by the choice (27) – (29) of $B^{(k)}$. So it remains to find a condition on the value of parameter ω, under which condition (c) is satisfied.

Let us define

$$\bar{\omega} = \min_{1 \leq i \leq m} \bar{\omega}_i, \tag{38}$$

where

$$\bar{\omega}_i = \min \left\{ \frac{1}{\theta_i}, \frac{3}{2+\theta_i} \right\}, \tag{39}$$

$$\theta_i = \frac{2}{\alpha_i} \sum_{j=1, j \neq i}^{m} |\alpha_{ij}|, \tag{40}$$

and $\alpha_{ij} = a_i^T Q^{-1} a_j$ for all $i, j = 1, \cdots, m$. Then we have the following lemma.

Lemma 3.2. *Let $B^{(k)}$ be given by (27) – (29). If ω satisfies*

$$0 < \omega < \bar{\omega}, \tag{41}$$

where $\bar{\omega}$ is defined by (38), then $\{B^{(k)} - C^{(k)}\}$ is uniformly positive definite.

Proof. Matrix M can be written

$$M = \hat{A} Q^{-1} \hat{A}^T = \begin{pmatrix} M_{11} & \cdots & M_{1m} \\ \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mm} \end{pmatrix}, \tag{42}$$

where

$$M_{ij} = \begin{pmatrix} \alpha_{ij} & -\alpha_{ij} \\ -\alpha_{ij} & \alpha_{ij} \end{pmatrix} \tag{43}$$

and $\alpha_{ij} = a_i^T Q^{-1} a_j$. (Note that $\alpha_{ii} = \alpha_i = a_i^T Q^{-1} a_i$.) So we may write

$$\begin{aligned} y^T \left(B^{(k)} - C^{(k)} \right) y &= y^T \left(2B^{(k)} - M \right) y \\ &= y^T \left(B^{(k)} + B^{(k)T} - M \right) y \\ &= y^T \left(\omega^{-1} \left(2D + K^{(k)} + K^{(k)T} \right) - M \right) y \\ &= y^T \Xi y, \end{aligned}$$

where Ξ is the $2m \times 2m$ matrix

$$\Xi = \begin{pmatrix} \Xi_1 & -M_{12} & \cdots & -M_{1m} \\ -M_{21} & \Xi_2 & \cdots & -M_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ -M_{m1} & -M_{m2} & \cdots & \Xi_m \end{pmatrix} \tag{44}$$

with

$$\begin{aligned} \Xi_i &= \omega^{-1}\left(2D_i + K_i^{(k)} + K_i^{(k)T}\right) - M_{ii} \\ &= \frac{\alpha_i}{\omega}\begin{pmatrix} 2-\omega & -1+\omega \\ -1+\omega & 2-\omega \end{pmatrix}. \end{aligned} \tag{45}$$

Since Ξ turns out to be independent of k, it suffices to show that it is positive definite. It is known (see [14, pages 48-50]) that any symmetric matrix is positive definite whenever it is strictly diagonally dominant and has positive diagonal elements. In the remainder of the proof, we show the latter properties under the condition $0 < \omega < \bar{\omega}$.

Observe that, by (39),

$$\bar{\omega}_i = \begin{cases} \dfrac{1}{\theta_i}, & \text{if } \theta_i \geq 1 \\ \dfrac{3}{2+\theta_i}, & \text{if } 0 \leq \theta_i < 1, \end{cases} \tag{46}$$

which implies $\bar{\omega}_i \leq 3/2$. Therefore we get $\bar{\omega} \leq 3/2$ by (38). Since $\omega < \bar{\omega}$, we see from (44) and (45) that the diagonal elements of Ξ are positive.

By the definition of Ξ_i and M_{ij}, matrix Ξ is strictly diagonally dominant if and only if

$$\begin{aligned} \frac{\alpha_i}{\omega}(2-\omega) &> \frac{\alpha_i}{\omega}|1-\omega| + 2\sum_{j=1, j\neq i}^{m} |\alpha_{ij}|, \\ &= \frac{\alpha_i}{\omega}|1-\omega| + \alpha_i\theta_i \end{aligned} \tag{47}$$

for all $i = 1, \cdots, m$, where the last equality follows from the definition (40) of θ_i. Let us define

$$\Psi_i(\omega) = \frac{1}{\omega}(2-\omega) - \frac{1}{\omega}|1-\omega| - \theta_i$$

for each i. Then, since $\alpha_i > 0$, (47) will be obtained if we show that $\Psi_i(\omega) > 0$.

First consider the case $\theta_i \geq 1$. By (46), we have $1/\bar{\omega}_i = \theta_i$ and $\bar{\omega}_i \leq 1$. Moreover, since $\omega < \bar{\omega}_i$, we have $\omega < 1$. Thus we get

$$\Psi_i(\omega) = 1/\omega - \theta_i > 1/\bar{\omega}_i - \theta_i = 0.$$

Next suppose that $0 \leq \theta_i < 1$. Then (46) implies $1/\bar{\omega}_i = (2 + \theta_i)/3$. Therefore, if $0 < \omega \leq 1$ then

$$\Psi_i(\omega) = 1/\omega - \theta_i > 1/\bar{\omega}_i - \theta_i = 2(1 - \theta_i)/3 > 0,$$

while, if $\omega > 1$ then

$$\Psi_i(\omega) = 3/\omega - 2 - \theta_i > 3/\bar{\omega}_i - 2 - \theta_i = 0.$$

Consequently, we get $\Psi_i(\omega) > 0$ for all $i = 1, \cdots, m$, indicating that matrix Ξ is strictly diagonally dominant. This completes the proof. $\square$

We have thus shown that the convergence conditions for Algorithm 3.2 are met if ω satisfies (41). Since equivalence of Algorithms 3.1 and 3.2 has already been established in Theorem 3.2, we obtain the following theorem.

Theorem 3.3. *Suppose that problem (1) is feasible. If parameter ω is chosen to satisfy (41), then the sequence $\{z^{(k)}\}$ generated by Algorithm 3.1 converges to a solution of the dual problem (5).*

3.3 Convergence of Algorithm 2.1

Finally we establish the following convergence theorem for Algorithm 2.1.

Theorem 3.4. *Let ω satisfy (41) and let $\{x^{(k)}\}$ be a sequence generated by Algorithm 2.1. If problem (1) is feasible, then $\{x^{(k)}\}$ converges to the unique solution of (1).*

Proof. By Theorem 3.1, it suffices to consider the sequence $\{x^{(k)}\}$ generated by Algorithm 3.1. Since (10) holds for all k and since $\{z^{(k)}\}$ converges to a solution z^* of problem (5) (cf. Theorem 3.3), the sequence $\{x^{(k)}\}$ converges to the point

$$x^* = x^0 - Q^{-1}\hat{A}^T z^*. \tag{48}$$

Since z^* satisfies the optimality conditions for problem (5), we have

$$\hat{A}Q^{-1}\hat{A}^T z^* + b - \hat{A}x^0 \geq 0, \tag{49}$$

$$z^* \geq 0, \tag{50}$$

$$\left(\hat{A}Q^{-1}\hat{A}^T z^* + b - \hat{A}x^0\right)^T z^* = 0. \tag{51}$$

By (48), we obtain

$$Q(x^* - x^0) + \hat{A}^T z^* = 0. \tag{52}$$

Moreover, (48) and (49) imply

$$b - \hat{A}x^* = b - \hat{A}x^0 + \hat{A}Q^{-1}\hat{A}^T z^* \geq 0, \tag{53}$$

and (48) and (51) yield

$$(b - \hat{A}x^*)^T z^* = (b - \hat{A}x^0 + \hat{A}Q^{-1}\hat{A}^T z^*)^T z^* = 0. \tag{54}$$

By the definitions (7) and (8) of $\hat{A}$ and b, respectively, (50) and (52)–(54) imply that x^* along with z^* satisfies the optimality conditions for problem (2), which is equivalent to problem (1). This completes the proof. □

4 Numerical Results

In this section, we present computational results for Algorithm 2.1 on a parallel computer called ADENA [13] at the Department of Applied Mathematics and Physics, Kyoto University. ADENA was originally designed to solve PDE's and consists of 16×16 PE array, i.e., 256 PEs. Its performance is estimated as 2.56 GFLOPS (peak) or 1.0 GFLOPS (effective). The programs used in the numerical experiments were coded in the parallel programming language ADETRAN [13], which is a modification of FORTRAN tailored to ADENA. All computations were done in double precision arithmetic.

We solved problem (1.1), in which Q is the identity matrix and A is a sparse matrix whose rows contain a certain fixed number of nonzero elements. The nonzero elements of A and the elements of δ were randomly chosen from the intervals $[-10, 10]$ and $[1, 10]$, respectively, and γ was set equal to $-\delta$. Note that any problem thus generated is feasible, because $\delta > 0$ and $\gamma < 0$ imply that the origin $x = 0$ satisfies all the constraints. All elements of the constant vector x^0 in the objective function were set to be 10. Closeness

of an iterate $x^{(k)}$ to the solution is measured in terms of the residual of the constraint inequalities of problem (1.1), which are defined by

$$r^{(k)} = \max_i \left\{ (a_i^T x^{(k)} - \delta_i)_+, (\gamma_i - a_i^T x^{(k)})_+ \right\}.$$

Because of the primal-dual nature of the algorithm, $x^{(k)}$ is optimal whenever it becomes feasible to problem (1.1), in which case $r^{(k)}$ vanishes.

First we applied Algorithm 2.1 to five test problems of the size $(n, m) = (200, 100)$, in which the density (i.e., the ratio of nonzero to all elements) in each row of A is fixed to be 10%, with several values of relaxation parameter ω. We observed that the algorithm was always convergent for sufficiently small ω, as demonstrated in Theorem 3.5. We observed that the proposed algorithm converged faster as ω became greater, but it failed to converge when ω was greater than a certain problem-dependent threshold value. Specifically, for those test problems, the algorithm did not converge for about a half of test problems when $\omega = 0.5$ and it completely failed to converge when $\omega \geq 0.6$. Figure 1 illustrates how $\{r^{(k)}\}$ decreases as the iteration proceeds, for the average of the five test problems. Moreover, the figure indicates that the algorithm is almost linearly convergent for any value of relaxation parameter ω, provided that it actually converges.

Next we also implemented a serial version of Algorithm 2.1 to compare it with parallel implementations. Programs for the serial version were coded in FORTRAN77 and run on a SUN4 SPARC station. By some preliminary experiments, we have found that the speed of SUN4 SPARC to perform floating point arithmetic operations is about 3/4 of the speed of a single processor element of the parallel computer ADENA. We made experiments with various sizes of test problems using the same termination criterion $r^{(k)} \leq 10^{-4}$ for the both versions. The results are summarized in Tables 1–4. In Tables 1 and 2, the density of A is 5% and in Tables 3 and 4, it is 10%. Note that the numbers in the columns "#Iter.", "Serial CPU" and "Parallel CPU" are the average over the five test problems. The column "Crude S/P ratio" shows the ratio of Serial CPU and Parallel CPU, which does not take into account the difference of the computers used. The column "Effective S/P ratio" shows the revised value of S/P ratio, which takes into account the difference of computer speeds. These tables indicate that the proposed algorithm is suited for parallel computation. Mainly because of inevitable communication overhead,

"Effective S/P ratio" cannot achieve the ideal theoretical ratio m. It may be observed that, since the amount of computation at one iteration becomes greater as a matrix A gets denser, the relative effect of communication overhead against the whole computation time becomes smaller (see [1, §1.3]). (Compare Table 1 and Table 3, and Table 2 and Table 4.)

Finally we compared the proposed algorithm with the SOR method proposed in [15] for solving interval constrainted problems. The SOR method sequentially executes m (minor) iterations at each (major) iteration, and each minor iteration acts upon a pair of inequalities in the constraints. Thus, we may consider that a single major iteration, i.e., m minor iterations, of the SOR method amounts to a single iteration of Algorithm 2.1. Note that the SOR algorithm of [15] also uses a relaxation parameter, which we denote here by ω^o. It is shown in [15] that the SOR algorithm is guaranteed to converge as long as ω^o is chosen to satisfy $0 < \omega^o < 2$. The SOR algorithm was run on the SUN4 SPARC station, while the proposed algorithm was run on the parallel computer ADENA. Tables 5 and 6 represent the results for the cases in which the densities of A are 5% and 10%, respectively. In each case, we tested three values of parameters ω and ω^o for both algorithms. In all the experiments, the termination criterion was $r^{(k)} \leq 10^{-4}$. The numbers in the tables represent the average over five problem instances for each problem size. From the tables, we may observe that the proposed algorithm with an appropriate value of ω is faster than the SOR algorithm, even if the difference of the used computers is taken into account.

5 Conclusion

In this paper, we have proposed a parallel relaxation algorithm (Algorithm 2.1) for solving the quadratic programming problem involving m pairs of inequality constraints. The proposed algorithm uses only m dual variables, each of which corresponds to a pair of inequality constraints. Thus the number of dual variables has been reduded by half in comparison with the straightforward application of a conventional relaxation method. In the course of proving the validity of the proposed algorithm, we have shown that it is equivalent to a relaxation algorithm that explicitly deals with $2m$ dual variables. In the

Table 1: Comparison[1] of serial and parallel implementations of the proposed algorithm for $\omega = 0.2$ with the density of A being 5%.

(n, m)	#Iter.	Serial CPU[2]	Parallel CPU[3]	Crude S/P ratio[4]	Effective S/P ratio[5]
(100,50)	4579	11.8	2.85	4.1	3.1
(150,50)	5529	19.4	3.52	5.5	4.1
(200,50)	8456	35.2	5.46	6.4	4.8
(150,100)	10716	74.3	6.84	10.9	8.2
(200,100)	9920	80.9	6.42	12.6	9.5
(250,100)	10786	109	7.17	15.2	11.4
(200,150)	15293	189	9.92	19.0	14.3
(250,150)	12697	192	8.41	22.8	17.1
(300,150)	13879	239	12.5	19.1	14.3

[1] Numbers are the average over five test problems for each problem size and the termination criterion is $r^{(k)} \leq 10^{-4}$.

[2] CPU sec by SUN4 SPARC station.

[3] CPU sec by ADENA.

[4] Crude ratio of Serial CPU sec and Parallel CPU sec, without taking into account the difference of the computers.

[5] Modified value of S/P ratio that takes into account the difference of the computers. (It is estimated that the speed of SUN4 SPARC station to perform floating point arithmetic operations is about 3/4 of the speed of a single processor element of the parallel computer ADENA.)

Table 2: Comparison[1] of serial and parallel implementations of the proposed algorithm for $\omega = 0.4$ with the density of A being 5%.

(n, m)	#Iter.	Serial CPU[2]	Parallel CPU[3]	Crude S/P ratio[4]	Effective S/P ratio[5]
(100,50)	2288	5.88	1.43	4.1	3.1
(150,50)	2762	9.70	1.76	5.5	4.1
(200,50)	4226	17.6	2.73	6.4	4.8
(150,100)	5357	37.2	3.42	10.9	8.2
(200,100)	4958	40.5	3.21	12.6	9.5
(250,100)	5390	54.3	3.57	15.2	11.4
(200,150)	7644	94.2	4.96	19.0	14.2
(250,150)	6346	96.0	4.21	22.8	17.1
(300,150)	6937	120	6.24	19.2	14.4

[1–5] See Table 1.

Table 3: Comparison[1] of serial and parallel implementations of the proposed algorithm for $\omega = 0.2$ with the density of A being 10%.

(n, m)	#Iter.	Serial CPU[2]	Parallel CPU[3]	Crude S/P ratio[4]	Effective S/P ratio[5]
(100,50)	8456	35.0	5.64	6.2	4.7
(150,50)	6201	35.8	4.26	8.4	6.3
(200,50)	3413	25.4	2.42	10.5	7.9
(150,100)	10530	118	7.28	16.3	12.2
(200,100)	12561	181	8.93	20.3	15.2
(250,100)	7630	138	5.60	24.7	18.5
(200,150)	19347	417	13.9	30.1	22.6
(250,150)	13983	368	10.4	35.5	26.6
(300,150)	15286	475	15.1	31.4	23.6

[1–5] See Table 1.

Table 4: Comparison[1] of serial and parallel implementations of the proposed algorithm for $\omega = 0.4$ with the density of A being 10%.

(n, m)	#Iter.	Serial CPU[2]	Parallel CPU[3]	Crude S/P ratio[4]	Effective S/P ratio[5]
(100,50)	423	17.5	2.82	6.2	4.7
(150,50)	3098	17.9	2.13	8.4	6.3
(200,50)	1704	12.7	1.21	10.5	7.9
(150,100)	5266	59.1	3.64	16.2	12.2
(200,100)	6278	90.4	4.47	20.2	15.2
(250,100)	3813	68.4	2.80	24.4	18.3
(200,150)	9671	209	6.93	30.1	22.6
(250,150)	6990	184	5.18	35.5	26.6
(300,150)	7641	238	7.58	31.3	23.5

[1–5] See Table 1.

Table 5: Comparison[1] with SOR algorithm for test problems with the density of A being 5%

	Proposed algorithm						SOR algorithm					
	$\omega = 0.3$		$\omega = 0.4$		$\omega = 0.5$		$\omega^o = 1.5$		$\omega^o = 1.7$		ω^o	
(n, m)	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	
(500,100)	4.89	5085	3.67	3813	2.93	3050	4.10	275	3.84	258	10.4	
(500,200)	8.00	8244	6.00	6182	4.80	4945	12.5	416	8.48	287	23.8	
(500,300)	19.1	14168	14.3	10625	11.6	8768	33.5	725	24.7	538	57.5	
(1000,100)	3.41	2219	2.55	1663	2.04	1329	3.76	136	5.39	196	16.4	
(1000,300)	29.9	14716	22.5	11036	18.0	8828	62.6	719	43.6	509	105	
(1000,500)	46.6	22282	35.0	16709	28.0	13366	172	1188	100	701	186	

[1] Numbers are the average over five test problems for each problem size and the termination criterion is $r^{(k)} \leq 10^{-4}$.

Table 6: Comparison[1] with SOR algorithm for test problems with the density of A being 10%

(n,m)	Proposed algorithm						SOR algorithm				
	$\omega = 0.3$		$\omega = 0.4$		$\omega = 0.5$		$\omega^o = 1.5$		$\omega^o = 1.7$		ω^o
	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec	#Iter.	CPU sec
(500,100)	2.39	2219	1.79	1663	1.43	1329	3.79	136	5.47	196	16.6
(500,200)	7.29	6678	5.46	5007	4.37	4005	17.5	310	19.0	336	51.6
(500,300)	23.1	14716	17.3	11036	13.8	8828	62.6	719	43.6	509	105
(1000,100)	1.19	674	0.89	504	0.71	402	3.30	59.4	5.69	104	18.1
(1000,300)	15.7	6459	11.7	4843	9.39	3874	46.8	283	52.4	315	149
(1000,500)	36.9	14067	27.7	10549	22.1	8438	210	736	141	502	351

[1] Numbers are the average over five test problems for each problem size and the termination criterion is $r^{(k)} \leq 10^{-4}$.

latter algorithm (Algorithm 3.1) , the updates of m pairs of dual variables are done in parallel and, for each pair, the two dual variables are updated sequentially but the order in which they are picked up may vary with iterations according to their current values. We have established the convergence of the proposed algorithm (Algorithm 2.1) by showing that the equivalent algorithm (Algorithm 3.1) belongs to a class of matrix splitting algorithms (Algorithm 3.2) for the linear complementarity problem and by examining the convergence conditions for Algorithm 3.2. We have also reported the results of some computational experience with the proposed algorithm.

Acknowledgement

The authors thank Paul Tseng for his helpful comments.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice Hall, New Jersey, 1989).

[2] Y. Censor, Row-action methods for huge and sparse systems and their applications, *SIAM Review* 23 (1981) 444-466.

[3] Y. Censor and A. Lent, An iterative row-action method for interval convex programming, *J. Opt. Theory Appl.* 34 (1981) 321-353.

[4] R.W. Cottle, J.-S. Pang and R.E. Stone, *The Linear Complementarity Problem* (Academic Press, San Diego, 1992).

[5] A.R. De Pierro and A.N. Iusem, On the convergence of SOR- and JOR-type methods for convex linear complementarity problems, *Lin. Alg. Appl.* 154-156 (1991) 601-614.

[6] G.T. Herman and A. Lent, A family of iterative quadratic optimization algorithms for pairs of inequalities, with application in diagnostic radiology, *Math. Prog. Study* 9 (1979) 15-29.

[7] C. Hildreth, A quadratic programming procedure, *Naval Res. Logistics Quart.* 4 (1957) 79-85.

[8] A.N. Iusem, On the convergence of iterative methods for symmetric linear complementarity problems, *Math. Prog.* 59 (1993) 33-48.

[9] A.N. Iusem and A.R. De Pierro, A simultaneous iterative method for computing projections on polyhedra, *SIAM J. Contr. Opt.* 25 (1987) 231-243.

[10] A.N. Iusem and S.A. Zenios, Interval underrelaxed Bregman's method and applications, University of Pennsylvania, Decision Science Department, Report 92-08-02, 1992.

[11] Z.-Q. Luo and P. Tseng, Error bound and convergence analysis of matrix splitting algorithms for the affine variational inequality problem, *SIAM J. Opt.* 2 (1992) 43-54.

[12] O.L. Mangasarian and R. De Leone, Parallel successive overrelaxation methods for symmetric linear complementarity problems and linear programs, *J Opt. Theory Appl.* 54 (1987) 437-446.

[13] T. Nogi, Parallel computation on ADENA, in: D.J. Evans, G.R. Joubert and H. Liddell, Eds., *Parallel Computing '91* (Elsevier, Amsterdam, 1992) 619-626.

[14] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, New York, 1970).

[15] Y. Shimazu, M. Fukushima and T. Ibaraki, A successive overrelaxation method for quadratic programming problems with interval constraints, *J. Oper. Res. Soc. Japan* 36 (1993).

[16] S.A. Zenios and Y. Censor, Massively parallel row-action algorithms for some nonlinear transportation problems, *SIAM J. Opt.* 1 (1991) 373-400.