Journal of the Operations Research
Society of Japan
Vol. 32, No. 1, March 1989

PRACTICAL POLYNOMIAL TIME ALGORITHMS FOR LINEAR COMPLEMENTARITY PROBLEMS

Shinji Mizuno Akiko Yoshise
Tokyo Institute of Technology

Takeshi Kikuchi
Yamaha Co., Ltd.

(Received April 18, 1988; Final September 26, 1988)

Abstract In this paper, we first propose three practical algorithms for linear complementarity problems, which are based on the polynomial time method of Kojima, Mizuno and Yoshise [5], and compare them by showing the computational complexities. Then we modify two of the algorithms in order to accelerate them. Through the computational experiments for three types of linear complementarity problems, we compare the proposed algorithms in practice and see the efficiency of the modified algorithms. We also estimate the practical computational complexity of each algorithm for each type of problems.

1. Introduction

Let M be an $n \times n$ matrix and $q \in R^n$, where R^n denotes the n-dimensional Euclidean space. The problem of finding an $(x, y) \in R^{2n}$ satisfying

$$(1) \quad y = Mx + q, \ < x, y >= 0, \ x \geq 0, \ y \geq 0,$$

or equivalently

$$(2) \quad y = Mx + q, \ x_i y_i = 0 \ (i = 1, 2, \ldots, n), \ x \geq 0, \ y \geq 0,$$

is known as a linear complementarity problem (abbreviated to LCP), where $< x, y >$ denotes the inner product of x and y. The LCP has various important applications

in linear and convex quadratic programs, bimatrix games and some other areas of engineering (Cottle and Dantzig [2], Lemke [7], Pang, Kaneko and Hallman [9], etc.). Throughout the paper, we shall restrict ourselves to the class of the LCPs with positive semi-definite matrices M. This class covers LCPs induced from linear and convex quadratic programs.

Several computational methods have been developed for solving LCPs. Most of the methods apply a sequence of pivoting operations to the system of linear equations $y = Mx + q$ (Cottle and Dantzig [2], Lemke [7], and Van der Heyden [10], etc.). In some worst cases, they require an exponential number of operations (Birge and Gana [1], Fathi [3], and Murty [8]). Recently Kojima, Mizuno and Yoshise [5] propose a polynomial time method for LCP. The method is an extension of new polynomial time methods for linear programming, which are originated by Karmarkar [4] and developed by many researchers. The method is theoretical and can hardly be implemented on a practical computer because the values of the initial point are too large ($2^{O(L)}$), the criteria of convergence are too small ($2^{-O(L)}$) and each arithmetic operation has to be done in $O(L)$ bits.

In this paper, we first propose three practical algorithms for linear complementarity problems, which are based on the polynomial time method of Kojima, Mizuno and Yoshise [5], and compare them by showing the computational complexities. Then we modify two of the algorithms in order to accelerate them. Through the computational experiments for three types of linear complementarity problems, we compare the proposed algorithms in practice and see the efficiency of the modified algorithms. We also estimate the practical computational complexity of each algorithm for each type of problems.

Chapter 2 describes the outline of the method of [5]. The method traces a path of centers from an initial point to a solution by generating a sequence of feasible points. In Chapter 3, we construct three algorithms, Algorithm A, Algorithm B and Algorithm C, which are based on the method of Chapter 2. It is shown that Algorithms A, B and C require at most $O(n^4L_1)$, $O(n^{3.5}L_1)$ and $O(n^3L_1)$ arithmetic operations, respectively, where L_1 is a number which depends on the initial point and the final point. Chapter 4 describes two modified algorithms, Algorithm A' and Algorithm B', which accelerate Algorithm A and Algorithm B, respectively. In Chapter 5, we show some computational results for three types of LCPs, Problem 1, Problem 2 and Problem 3. Using the computational results, we estimate the practical computational complexity of each algorithm for each type of LCPs. We also compare Algorithms A, B and C in practice and see the differences between Algorithms A and A' and between Algorithms B and B'. Chapter 6 gives the conclusions.

2. Tracing Method of a Path of Centers

Here we show the basic idea of Kojima, Mizuno and Yoshise [5] on which we construct our algorithms in Chapter 3. They employ the symbols S for the set of all the feasible solutions of LCP and S_{int} for its interior points;

$$\begin{aligned} S &= \{(x,y):\ y = Mx+q,\ x \geq 0,\ y \geq 0\}, \\ S_{int} &= \{(x,y):\ y = Mx+q,\ x > 0,\ y > 0\}. \end{aligned}$$

Then they introduce the map H;

$$H(\mu,x,y) = (XYe - \mu e, y - Mx - q) \text{ for } \mu \geq 0,\ x \geq 0,\ y \geq 0,$$

where $X = \mathrm{diag}(x_i)$, which is the $n \times n$ diagonal matrix with diagonal elements x_i $(i = 1,2,\ldots,n)$, $Y = \mathrm{diag}(y_i)$ and e the n-dimensional vector of ones. Then the LCP (1) is expressed as

(3) $H(0,x,y) = 0,\ x \geq 0,\ y \geq 0.$

For each $\mu \geq 0$, they consider the system of equations

(4) $H(\mu,x,y) = 0,\ x \geq 0,\ y \geq 0.$

For each $\mu > 0$, the system (4) has a unique solution which is called a center (of the feasible region S) and denoted by $(x(\mu), y(\mu))$. They show that the set of centers

$$\begin{aligned} S_{cen} &= \{(x(\mu), y(\mu)):\ \mu > 0\} \\ &= \{(x,y):\ y = Mx+q,\ XYe = \mu e,\ x > 0,\ y > 0,\ \mu > 0\} \end{aligned}$$

forms a smooth curve in the set S_{int}. So we call S_{cen} the path of centers.

Since the system (4) for $\mu = 0$ is equivalent to the system (1), the center $(x(\mu), y(\mu))$ for a sufficiently small $\mu > 0$ is close enough to a solution of LCP. In other word, the path S_{cen} runs through the interior S_{int} to a solution of the LCP. Hence an approximate solution (x,y) of LCP will be found by tracing the path S_{cen} from an initial point until we attain $< x,y > \leq \epsilon$ for a sufficiently small $\epsilon > 0$.

Now we shall show the method for tracing the path of centers S_{cen}. Let N be a neighborhood of S_{cen} in S_{int};

$$S_{cen} \subset N \subset S_{int}.$$

At the present stage, we assume that an initial point (x^0, y^0) in N is known. Starting from the initial point, we generates a sequence $\{(x^k, y^k)\} \subset N$ by repeating the following procedure for $k = 0, 1, \ldots$ until we get an approximate solution (x^k, y^k) which satisfies $< x^k, y^k > \leq \epsilon$.

Procedure which generates output (x^{k+1}, y^{k+1}) from input (x^k, y^k):

Step 1: Determine the parameter μ^k.
Step 2: Calculate the direction $(\Delta x, \Delta y)$ at (x^k, y^k) for finding the solution of the system $H(\mu^k, x, y) = 0$.
Step 3: Compute the step size $t \in [0, 1]$ and the next point (x^{k+1}, y^{k+1}) such that

$$(x^{k+1}, y^{k+1}) = (x^k, y^k) - t(\Delta x, \Delta y) \in N.$$

3. Three Polynomial Time Algorithms

In the previous chapter, we show the method for tracing the path of centers S_{cen}. There are five problems in the method:

(a) How to define a neighborhood N of S_{cen}.
(b) How to find an initial point in N.
(c) How to determine the parameter μ^k at (x^k, y^k).
(d) How to calculate the direction $(\Delta x, \Delta y)$ at (x^k, y^k).
(e) How to get the step size t.

In this chapter, we contruct three algorithms, Algorithm A, Algorithm B and Algorithm C, by providing solutions to the above five problems.

3.1. Algorithm A

We can regard Algorithm A as an extension of the $O(n^4L)$ method proposed by Kojima, Mizuno and Yoshise [6] for linear programming.

(a) For a contant $\pi \geq 1$, we define the neighborhood $N_1(\pi)$ of S_{cen} by

$$N_1(\pi) = \{(x, y) \in S_{int} : \ f_{ave} \leq \pi f_{min}\},$$

where

$$f_{ave} = < x, y > /n \text{ and } f_{min} = \min_i x_i y_i.$$

Since $x_i(\mu)y_i(\mu) = \mu$ for each $(x(\mu), y(\mu)) \in S_{cen}$, we have $f_{ave} = f_{min} = \mu$ for $(x(\mu), y(\mu)) \in S_{cen}$ which implies $S_{cen} \subset N_1(\pi)$.

(b) In order to find an initial point in $N_1(\pi)$, we contruct an artificial LCP of the form (1). Let the original LCP, which we want to solve, be

$$(5) \quad \bar{y} = \bar{M}\bar{x} + \bar{q}, \quad < \bar{x}, \bar{y} >= 0, \quad \bar{x} \geq 0, \quad \bar{y} \geq 0,$$

where $\bar{x} \in R^m$, $\bar{y} \in R^m$, $\bar{q} \in R^m$ and $\bar{M}$ is a positive semi-definite $m \times m$ matrix. Then we define an artificial LCP by

$$(6) \quad y = Mx + q, \quad < x, y >= 0, \quad x \geq 0, \quad y \geq 0,$$

for

$$(7) \quad y = \begin{pmatrix} \bar{y} \\ y_n \end{pmatrix}, \quad x = \begin{pmatrix} \bar{x} \\ x_n \end{pmatrix}, \quad q = \begin{pmatrix} \bar{q} \\ q_n \end{pmatrix}, \quad M = \begin{pmatrix} \bar{M} & \bar{e} \\ -\bar{e}^T & 0 \end{pmatrix},$$

where $n = m + 1$, $\bar{e}$ is the m-dimensional vector of ons and q_n is a contant. We easily see that the above $n \times n$ matrix M is positive semi-definite and (6) has the same form as (1). The next theorem shows that the LCP (5) can be solved, from a practical point of view, by obtaining a solution of LCP (6) for a sufficiently large q_n.

Theorem 3.1. *For any solution* $(x^*, y^*) = (\bar{x}^*, x_n^*, \bar{y}^*, y_n^*)$ *of (6),*

(i) *if* $x_n^* = 0$ *then* $(\bar{x}^*, \bar{y}^*)$ *is a solution of (5),*
(ii) *if* $x_n^* > 0$ *then the LCP (5) has no solution in the set* $\{(\bar{x}, \bar{y}) :< \bar{e}, \bar{x} >< q_n\}$.

Proof: From (6) and (7), we have

$$\bar{y}^* = \bar{M}\bar{x}^* + x_n^*\bar{e} + \bar{q}, \quad y_n^* = q_n - < \bar{e}, \bar{x}^* >, < \bar{x}^*, \bar{y}^* >= 0, \quad x_n^* y_n^* = 0.$$

Hence we easily see (i). Now suppose that $x_n^* > 0$. Then we have $y_n^* = 0$, which implies

$$(8) \quad q_n =< \bar{e}, \bar{x}^* > .$$

Let $(\bar{x}', \bar{y}')$ be any solution of (5) then

$$\begin{aligned} 0 &\leq <\bar{x}' - \bar{x}^*, \bar{M}(\bar{x}' - \bar{x}^*)> \\ &= <\bar{x}' - \bar{x}^*, (\bar{y}' - \bar{q}) - (\bar{y}^* - x_n^*\bar{e} - \bar{q})> \\ &= <\bar{x}' - \bar{x}^*, \bar{y}' - \bar{y}^*> + x_n^* <\bar{x}' - \bar{x}^*, \bar{e}> \\ (9) \quad &\leq x_n^* <\bar{x}' - \bar{x}^*, \bar{e}>, \end{aligned}$$

where the last inequality follows from $<\bar{x}', \bar{y}'> = 0$, $<\bar{x}^*, \bar{y}^*> = 0$ and $\bar{x}', \bar{y}', \bar{x}^*, \bar{y}^* \geq 0$. From (8), (9) and $x_n^* > 0$, we have $<\bar{e}, \bar{x}'> \geq q_n$. ∎

The next theorem gives an initial point of Algorithm A.

Theorem 3.2. *For contants $\pi > 1$ and $\xi > 0$, let*

$$(10) \quad q_n = n\xi, \quad u = \begin{pmatrix} \xi\bar{M}\bar{e} + \bar{q} \\ 0 \end{pmatrix},$$

$$(11) \quad u_{min} = \min_i u_i, \quad u_{ave} = <e, u> / n = <\bar{e}, \xi\bar{M}\bar{e} + \bar{q}> / n,$$

$$(12) \quad \eta_1 = \max\{1 - u_{min}, (u_{ave} - \pi u_{min})/(\pi - 1)\}.$$

Then the following point (x, y) belongs to $N_1(\pi)$:

$$(13) \quad y = \begin{pmatrix} \bar{y} \\ y_n \end{pmatrix} = \begin{pmatrix} \eta_1\bar{e} + \xi\bar{M}\bar{e} + \bar{q} \\ \xi \end{pmatrix}, \quad x = \begin{pmatrix} \bar{x} \\ x_n \end{pmatrix} = \begin{pmatrix} \xi\bar{e} \\ \eta_1 \end{pmatrix}.$$

Proof: We easily see that $x > 0$, $y > 0$ and $y = Mx + q$. So we only show that $f_{ave} \leq \pi f_{min}$. From the above definition, we have

$$\begin{aligned} f_{ave} &= <x, y> / n \\ &= (<\xi\bar{e}, \eta_1\bar{e} + \xi\bar{M}\bar{e} + \bar{q}> + \eta_1\xi)/n \\ &= \xi(\eta_1 + u_{ave}), \\ f_{min} &= \min_i x_i y_i = \min_i \xi(\eta_1 + u_i) \\ &= \xi(\eta_1 + u_{min}). \end{aligned}$$

Hence we have

$$\pi f_{min} - f_{ave} = \xi\{(\pi - 1)\eta_1 + (\pi u_{min} - u_{ave})\} \geq 0.$$

(c) Let $\sigma \in (0,1)$ be a contant. For k-th point (x^k, y^k), we set $\mu^k = \sigma < x^k, y^k >$.

(d) We calculate the direction $(\Delta x, \Delta y)$ by Newton Method for the system of equations $H(\mu^k, x, y) = 0$ at the point (x^k, y^k). Let $X = \mathrm{diag}(x_i^k)$ and $Y = \mathrm{diag}(y_i^k)$ then the Newton direction $(\Delta x, \Delta y)$ is expressed as (see [5])

$$\Delta x = (M + X^{-1}Y)^{-1}(Ye - \mu^k X^{-1}e) \text{ and } \Delta y = M\Delta x. \tag{14}$$

(e) We calculate the maximum step size t which satisfies

$$(x^k, y^k) - t(\Delta x, \Delta y) \in N_1(\pi).$$

This condition is equivalent to

$$< x^k - t\Delta x, y^k - t\Delta y > /n \leq \pi(x_i^k - \Delta x_i)(y_i^k - \Delta y_i) \text{ for } i = 1, 2, \ldots, n.$$

Since this system conists of n quadratic inequalities, we can easily get the maximum value of t by solving n quadratic equations.

3.2. Algorithm B

Algorithm B is the same as the $O(n^{3.5}L)$ method proposed by Kojima, Mizuno and Yoshise [5] except for the artificial problem, the initial point and the criteria of convergence. The method of [5] uses an artificial LCP which is equivalent to the original LCP and computes an exact solution. On the other hand, Algorithm B uses an artificial LCP which is related to the original LCP by Theorem 3.1 and computes an approximate solution.

(a) For a contant $\alpha \in (0, 0.1]$, we define a neighborhood $N_2(\alpha)$ of S_{cen} by

$$N_2(\alpha) = \{(x, y) \in S_{int} : \ \|XYe - f_{ave}e\| \leq \alpha f_{ave}\}.$$

This neighborhood is used in [5] and called an α-center neighborhood of S_{cen}.

(b) In the same way as Algorithm A, we take the artificial LCP (6) and (7) for the original LCP (5). The next theorem gives an initial point.

Theorem 3.3. *For contants $\alpha \in (0, 0.1]$ and $\xi > 0$, define q_n, u, u_{min} and u_{ave} by (10) and (11). Let*

$$\eta_2 = \max\{1 - u_{min}, \|u - u_{ave}e\|/\alpha - u_{ave}\}, \tag{15}$$

then the following point (x, y) belongs to $N_2(\alpha)$;

$$y = \begin{pmatrix} \bar{y} \\ y_n \end{pmatrix} = \begin{pmatrix} \eta_2\bar{e} + \xi\bar{M}\bar{e} + \bar{q} \\ \xi \end{pmatrix}, \quad x = \begin{pmatrix} \bar{x} \\ x_n \end{pmatrix} = \begin{pmatrix} \xi\bar{e} \\ \eta_2 \end{pmatrix} \tag{16}$$

Proof: It will be verified by simple calculation. ∎

(c) Let $\delta = \alpha/(1-\alpha)$. For k-th point (x^k, y^k), we set

$$\mu^k = (1 - \delta/n^{0.5}) < x^k, y^k > .$$

(d) We calculate the direction $(\Delta x, \Delta y)$ by (14).

(e) The step size t is fixed to 1, because the next point $(x^{k+1}, y^{k+1}) = (x^k, y^k) - (\Delta x, \Delta y)$ always belongs to $N_2(\alpha)$ whenever $\alpha \in (0, 0.1]$ (see [5]).

3.3. Algorithm C

Algorithm C is the same as Algorithm B except for (d). We roughly show how to calculate the direction $(\Delta x, \Delta y)$ (see [5] for the detail). For a contant $\beta \in (0, 1)$, we have an approximation $(\tilde{x}^k, \tilde{y}^k)$ of (x^k, y^k) at each k-th iteration as follows;

$$\begin{aligned} \tilde{x}_i^k/x_i^k &\in [1/(1+\beta), 1+\beta], \text{ for } i = 1, 2, \ldots, n, \\ \tilde{y}_i^k/y_i^k &\in [1/(1+\beta), 1+\beta], \text{ for } i = 1, 2, \ldots, n. \end{aligned}$$

Suppose that we have $\tilde{x}^{k-1}$, $\tilde{y}^{k-1}$ and the matrix $(M + \tilde{X}^{-1}\tilde{Y})^{-1}$ for $\tilde{X} = \mathrm{diag}(\tilde{x}_i^{k-1})$ and $\tilde{Y} = \mathrm{diag}(\tilde{y}_i^{k-1})$ at $(k-1)$-th iteration. At k-th iteration, we get the direction $(\Delta x, \Delta y)$ by the next procedure:

Step 1: Set $\tilde{x}^k = \tilde{x}^{k-1}$, $\tilde{y}^k = \tilde{y}^{k-1}$.

Step 2: For $i = 1, 2, \ldots, n$, if $\tilde{x}_i^k/x_i^k \notin [1/(1+\beta), 1+\beta]$ or $\tilde{y}_i^k/y_i^k \notin [1/(1+\beta), 1+\beta]$, then reset $\tilde{x}_i^k = x_i^k$ and $\tilde{y}_i^k = y_i^k$ and compute $(M + \tilde{X}^{-1}\tilde{Y})^{-1}$ for new $\tilde{X} = \mathrm{diag}(\tilde{x}_i^k)$ and $\tilde{Y} = \mathrm{diag}(\tilde{y}_i^k)$ by using the rank on update (for example, on pivoting operation).

Step 3: Compute $(\Delta x, \Delta y)$ by $\Delta x = (M + \tilde{X}^{-1}\tilde{Y})^{-1}(\tilde{Y}e - \mu^k \tilde{X}^{-1}e)$ and $\Delta y = M\Delta x$.

3.4. Computational complexities of the three algorithms

In this section, we show the computational complexities of Algorithms A, B and C. In the same way as the case of linear programming ([6]), we can prove that the sequence $\{(x^k, y^k)\}$ generated by Algorithm A satisfies

$$< x^k, y^k > \leq (1 - \delta_1/n)^k < x^0, y^0 >$$

for a contant $\delta_1 > 0$. This implies that an approximate solution (x^k, y^k), which satisfies $< x^k, y^k > \leq \epsilon$, is found after at most $O(nL_1)$ iterations for $L_1 = \log(< x^0, y^0 > /\epsilon)$. Since each iteration involves $O(n^3)$ arithmetic operations for the computation of the direction $(\Delta x, \Delta y)$, Algorithm A requires $O(n^4 L_1)$ arithmetic operations in total. In the same way as Kojima, Mizuno and Yoshise [5], we can show that Algorithm B converges in $O(n^{0.5}L_1)$ iterations with a computational effort of $O(n^3)$ arithmetic operations per iteration. Hence Algorithm B requires $O(n^{3.5}L_1)$ arithmetic operations in total. We can also show that Algorithm C converges in $O(n^{0.5}L_1)$ iteration with an average number of $O(n^{2.5})$ arithmetic operations per iteration. Thus Algorithm C requires $O(n^3 L_1)$ arithmetic operations in total.

4. Two Modified Algorithms

In this chapter, we propose Algorithm A' and Algorithm B' which are modifications of Algorithm A and Algorithm B, respectively.

4.1. Algorithm A'

In Algorithm A, we always set $\mu^k = \sigma < x^k, y^k >$. In Algorithm A', we take the parameter μ^k less than or equal to $\sigma < x^k, y^k >$. When the step size t equals to 1 in Step 3 of the procedure given in Chapter 2, the next point (x^{k+1}, y^{k+1}) is an approximate solution of $(x(\mu^k), y(\mu^k))$. Since $< x(\mu^k), y(\mu^k) >= n\mu^k$, the value $< x^{k+1}, y^{k+1} >$ will roughly be proportional to μ^k when $t = 1$. Hence we will be able to get a good next point by taking μ^k as small as possible when we can take the step size $t = 1$.

From (14), we can regard the direction $(\Delta x, \Delta y)$ as a function of μ^k as follows;

$$(\Delta x, \Delta y) = (\Delta x', \Delta y') - \mu^k(\Delta x", \Delta y"),$$

where

$$\begin{array}{lllll} \Delta x' & = & (M+X^{-1}Y)^{-1}Ye, & \Delta y' & = & M\Delta x', \\ \Delta x" & = & (M+X^{-1}Y)^{-1}X^{-1}e, & \Delta y" & = & M\Delta x". \end{array}$$

At the k-th iteration of Algorithm A', we first compute the directions $(\Delta x', \Delta y')$ and $(\Delta x", \Delta y")$ then compute $(\Delta x, \Delta y)$ for $\mu^k = \sigma < x^k, y^k >$. If

$$(x^k, y^k) - (\Delta x, \Delta y) \notin N_1(\pi),$$

we determine the step size t and the next point (x^{k+1}, y^{k+1}) by the same way as in Algorithm A. Otherwise the value of μ^k is changed to the minimum value $\mu' \in [0, \sigma < x^k, y^k >]$ which satisfies

$$(x^k, y^k) - (\Delta x', \Delta y') + \mu'(\Delta x", \Delta y") \in N_1(\pi),$$

and compute the next point by

$$(x^{k+1}, y^{k+1}) = (x^k, y^k) - (\Delta x', \Delta y') + \mu^k(\Delta x", \Delta y"). \tag{17}$$

4.2. Algorithm B'

Algorithm B' is the same as Algorithm B except for the computation of μ^k and the next point. In Algorithm B', we set μ^k to the minimum value $\mu' \in [0, 1]$ which satisfies

$$(x^k, y^k) - (\Delta x', \Delta y') + \mu'(\Delta x", \Delta y") \in N_2(\alpha),$$

and compute the next point by (17). Ye [11] proposes this type of modification for quadratic programming.

5. Computational Results

In this chapter, we show some computational results for three types of LCPs. We use C language (64 bits real number and 32 bits integer number) for programming the algorithms and implement on SUN-3 computer (UNIX operating system). The three types of LCPs are:

Problem 1. The linear complementarity problem (5) where

$$\bar{M} = \begin{pmatrix} 1 & 2 & 2 & \cdots & 2 \\ 0 & 1 & 2 & \cdots & 2 \\ 0 & 0 & 1 & \cdots & 2 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \text{ and } \bar{q} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}.$$

Problem 2. The linear complementarity problem (5) where

$$\bar{M} = \begin{pmatrix} 1 & 2 & 2 & \cdots & 2 \\ 2 & 5 & 6 & \cdots & 6 \\ 2 & 6 & 9 & \cdots & 10 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 2 & 6 & 10 & \cdots & 4(m-1)+1 \end{pmatrix} \text{ and } \bar{q} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}.$$

Problem 3. The linear complementarity problem (5) where

$\bar{M} = A^T A$ for an $m \times m$ matrix A whose elements a_{ij} $(i, j = 1, 2, \ldots, m)$ are random numbers in $(-1, 1)$

and

$\bar{q}$ is an m-dimensional vector of random numbers $\bar{q}_i$ $(i = 1, 2, \ldots, m)$ in $(-1, 1)$.

Note that all the matrices $\bar{M}$ appearing in the above problems are positive semi-definite. Murty [8] and Fathi [3] show that popular complementarity pivoting methods [2] [7] require 2^m steps to solve Problem 1 or Problem 2. We use Problems 1 and 2 of each size $m = 8, 16, 32, 64$ and 128. As for Problem 3, we use 10 examples of each size $m = 8, 16, 32$ and 64 and one example of the size $m = 128$. For each LCP of the form (5), we solve the artificial problem (6) by taking the initial point (13) for Algorithms A and A' or (16) for Algorithms B, C and B'. In each case, we use the following constants

$$\xi = 10^6, \quad \epsilon = 10^{-6}, \quad \pi = 2, \quad \sigma = 0.5, \quad \alpha = 0.1.$$

The computational results are given in Table 1 for Problem 1, Table 2 for Problem 2 and Table 3 for Problem 3, where

(a) the row L shows the value of $\log_{10}(< x^0, y^0 > / < x^k, y^k >)$ for the initial point (x^0, y^0) and the terminated point (x^k, y^k) such that $< x^k, y^k > \leq \epsilon$,

Table 1. Computational results for Problem 1

Alg.		m = 8	m = 16	m = 32	m = 64	m = 128
A	L	20.33	20.82	21.37	21.92	22.57
	Ite.	77	81	86	92	100
	CPU	0.517	1.650	5.767	22.93	94.73
B	L	20.85	21.45	22.02	22.64	23.26
	Ite.	112	166	245	361	530
	CPU	0.700	3.167	16.10	87.72	499.6
C	L	20.85	21.45	22.02	22.62	23.22
	Ite.	112	166	250	370	548
	CPU	3.700	31.62	244.8	2077.	13720.
A'	L	22.77	22.14	22.50	22.85	24.20
	Ite.	37	40	44	49	56
	CPU	0.717	1.783	5.367	19.30	78.60
B'	L	21.40	21.79	22.04	25.50	25.33
	Ite.	28	29	30	32	33
	CPU	0.967	2.117	5.333	16.28	53.88

Table 2. Computational results for Problem 2

Alg.		m = 8	m = 16	m = 32	m = 64	m = 128
A	L	21.43	22.16	22.98	24.01	24.87
	Ite.	81	85	90	96	102
	CPU	1.033	5.017	31.33	222.8	1746.
B	L	21.89	22.78	23.64	24.57	25.48
	Ite.	118	177	264	393	582
	CPU	1.400	10.22	89.57	914.7	10110.
C	L	21.89	22.78	25.70	24.56	25.47
	Ite.	118	177	270	404	605
	CPU	3.983	34.15	259.3	2135.	14990.
A'	L	24.29	22.70	25.87	24.69	25.72
	Ite.	41	44	49	53	58
	CPU	1.100	3.883	20.62	136.4	1064.
B'	L	24.67	25.43	26.54	27.35	25.58
	Ite.	32	34	36	38	39
	CPU	1.317	3.917	17.13	101.6	713.1

Table 3. Computational results for Problem 3

Alg.			m = 8	m = 16	m = 32	m = 64	m = 128
A	L	min.	19.55	20.00	21.25	22.02	23.16
		ave.	19.90	20.69	21.54	22.23	
		max.	20.08	20.99	21.72	22.38	
	Ite.	min.	74	79	84	90	99
		ave.	74.9	79.5	84.7	90.5	
		max.	76	80	85	91	
	CPU	min.	0.917	4.683	29.30	208.9	1739.
		ave.	0.977	4.800	29.79	212.9	
		max.	1.050	5.050	30.18	217.6	
B	L	min.	20.27	20.98	21.76	22.51	23.47
		ave.	20.51	21.18	21.89	22.62	
		max.	20.71	21.41	22.03	22.70	
	Ite.	min.	109	162	242	359	535
		ave.	110.1	164.1	243.4	360.7	
		max.	112	166	245	362	
	CPU	min.	1.283	9.400	82.93	836.8	9712.
		ave.	1.362	9.465	83.59	864.1	
		max.	1.400	9.567	84.43	980.8	
C	L	min.	20.27	20.98	21.71	22.49	23.47
		ave.	20.51	21.21	21.87	22.66	
		max.	20.71	21.41	21.97	22.69	
	Ite.	min.	109	162	246	368	554
		ave.	110.1	164.1	247.8	370.1	
		max.	112	166	249	371	
	CPU	min.	3.633	30.77	245.7	2084.	13780.
		ave.	3.885	31.17	248.8	2110.	
		max.	4.430	31.53	250.8	2124.	
A'	L	min.	20.46	20.56	21.31	22.30	23.97
		ave.	21.72	22.03	22.22	22.74	
		max.	22.66	23.77	23.75	23.81	
	Ite.	min.	36	40	44	50	58
		ave.	37.4	42.2	45.9	51.4	
		max.	40	43	48	52	
	CPU	min.	0.950	3.600	18.63	127.2	1053.
		ave.	1.010	3.818	19.66	132.8	
		max.	1.133	3.983	20.53	136.1	
B'	L	min.	20.49	22.02	21.83	22.54	24.42
		ave.	21.76	22.53	22.92	23.08	
		max.	22.95	24.39	24.17	23.77	
	Ite.	min.	27	31	32	35	37
		ave.	29.3	32.1	33.5	35.5	
		max.	30	33	35	36	
	CPU	min.	1.117	3.483	15.15	93.33	678.9
		ave.	1.202	3.688	16.00	94.91	
		max.	1.267	3.810	16.58	96.33	

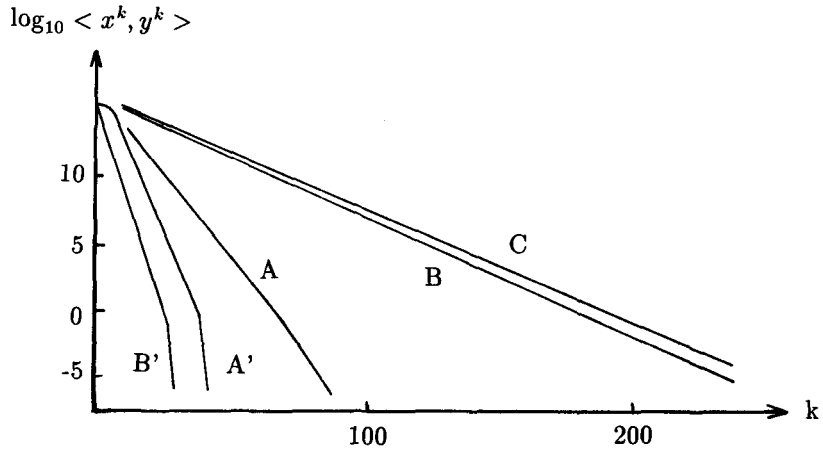

Fig.1 Graph of $(k, \log_{10} < x^k, y^k >)$ for Problem 1

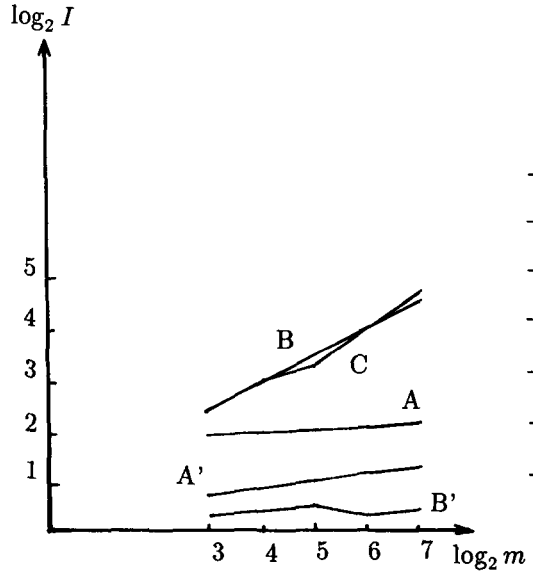

Fig.2 Graph of $(\log_2 m, \log_2 I)$ for Problem 1

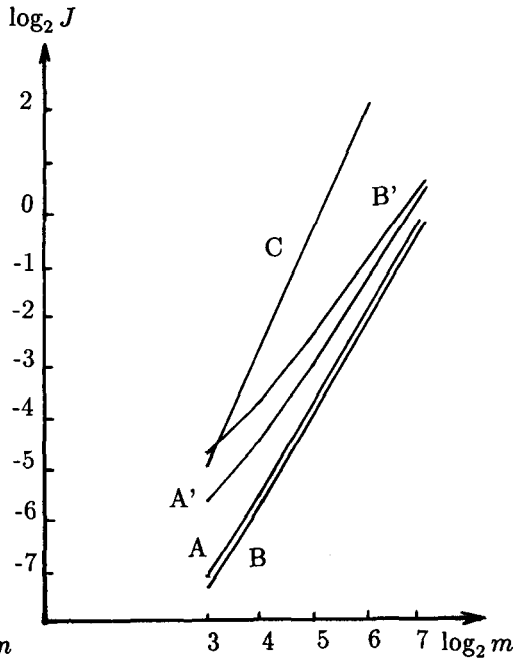

Fig.3 Graph of $(\log_2 m, \log_2 J)$ for problem 1

(b) the row *Ite.* shows the iteration number k of the terminated point (x^k, y^k),
(c) the row *CPU* shows the CPU time by second from when we get the initial point until when we get the terminated point,
(d) the row *min.*, *ave.* and *max.* in Table 3 show the minimum, average and maximum values of each data item for 10 examples, respectively.

In each case, an approximate solution is obtained. From Table 1, Table 2 and Table 3, we can see that

(1) Algorithm A is faster than Algorithm B and Algorithm B is faster than Algorithm C,
(2) Algorithm A' accelerates Algorithm A and Algorithm B' accelerates Algorithm B except for small problems.
(3) In Table 3, the difference between minimum and maximum values for each case is very small. So each algorithm is stable with respect to computation time and the number of iterations.

In order to see the way of convergence, we get the data of the values of all $< x^k, y^k >$ for Problem 1 of $m = 32$ and illustrate the graph of $(k, \log_{10} < x^k, y^k >)$ in Fig.1. We can observe from Fig.1 that

(4) the convergence of each of Algorithms A, B and C is linear,
(5) the convergence of each of Algorithms A' and B' is at least linear and is superlinear when the values $< x^k, y^k >$ become sufficiently small.

Though we here omit the figures for Problem 2 and Problem 3, we get the same observations as (4) and (5).

We see in section 3.4 that the iteration number is at most $O(n^c L_1)$ for $c = 1$ (Algorithm A) or $c = 0.5$ (Algorithm B and Algorithm C), and the number of arithmetic operations per iteration is at most $O(n^d)$ for $d = 3$ (Algorithm A and Algorithm B) or $d = 2.5$ (Algorithm C). So we consider that the following values

$$I = (\text{data in } \textit{Ite.})/(\text{data in } L)$$

and

$$J = (\text{data in } \textit{CPU})/(\text{data in } \textit{Ite.})$$

will be in proportional to some exponential of the problem size m for each problem and each algorithm, i.e., there are c and d such that

I is proportional to m^c

and

J is proportional to m^d.

Then we say that the practical computational complexity of iteration numbers is $O(m^c L_1)$, the practical computational complexity per iterations is $O(m^d)$ and the practical computational complexity is $O(m^{c+d} L_1)$. In order to guess the values of c and d for Problem 1, we draw the graph of $(\log_2 m, \log_2 I)$ in Fig.2 and the graph of $(\log_2 m, \log_2 J)$ in Fig.3. It will be found that the data for each algorithm almost lay on a straight line in each of Fig.2 and Fig.3. So we estimate the values of c and d as the gradients of regression lines of the data. In the same way, we get the estimate values of c and d for Problem 2 and Problem 3. We show the obtained estimate values in Table 4, where we also put the theoretical upper bounds which are given in Section 3.4.

Table 4. The estimation of c and d

		Alg. A	Alg. B	Alg. C	Alg. A'	Alg. B'
Theory	c	1	0.5	0.5	1	0.5
	d	3	3	2.5	3	3
	c + d	4	3.5	3	4	3.5
Problem 1	c	0.05	0.52	0.54	0.13	- 0.01
	d	1.79	1.81	2.40	1.55	1.40
	c + d	1.84	2.33	2.94	1.68	1.39
Problem 2	c	0.03	0.52	0.54	0.10	0.05
	d	2.61	2.64	2.38	2.51	2.26
	c + d	2.64	3.16	2.92	2.61	2.31
Problem 3	c	0.04	0.52	0.54	0.13	0.06
	d	2.54	2.56	2.43	2.24	2.06
	c + d	2.58	3.08	2.97	2.37	2.12

From Table 4, we can see that

(6) The values of c for each algorithm are almost the same in Problem 1, Problem 2 and Problem 3. However the values of d for each algorithm except for Algorithm C are less in Problem 1 than in Problem 2 and Problem 3 (probably because the matrix M of Problem 1 is upper triangular).

(7) The practical computational complexities of Algorithm C are almost the same as the theoretical upper bounds, but those of the other algorithms are less than the theoretical upper bounds.

(8) The values of c in Algorithms A, A' and B' are almost zero. This fact implies that the iteration number is proportional only to L_1 but not to m.

(9) The practical computational complexity of Algorithm A is less than those of Algorithm B and Algorithm C for each problem.

(10) The practical computational complexity of Algorithm B' is less than that of Algorithm B for each problem.

6. Conclusions

This paper describes three prototype algorithms (Algorithm A, Algorithm B and Algorithm C) and two modified algorithms (Algorithm A' and Algorithm B') for linear complementarity problems. Then we compare the algorithms in theory (Section 3.4) and in practice (Chapter 5). We can conclude that all proposed algorithms are polynomial time in theory and in practice, and

Algorithm C	>	Algorithm B	>	Algorithm A	in theory,
Algorithm A	>	Algorithm B	>	Algorithm C	in practice,
Algorithm A'	>	Algorithm A			in practice,
Algorithm B'	>	Algorithm B			in practice,

where ">" means "better than" or "faster than".

Acknowledgment

The authors wish to thank Professor Masakazu Kojima for his useful suggestions, Professor Masao Mori for his warm encouragement to their study and Mr. Isao Kohiyama for his help in our computational experiments. They are also grateful to Professor Yoshitsugu Yamamoto for his comment on Theorem 2 and referees for their helpful comments. Part of the research of the first author was supported by Grant-in-Aid for Encouragement of Young Scientists 63731004 of the Ministry of Education, Science and Culture.

References

[1] J. R. Birge and A. Gana. Computational complexity of Vander Heyden's variable dimensional algorithm and Dantzig-Cottle's principal pivoting method for solving LCP's. *Mathematical Programming*, 26:316–325, 1983.

[2] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[3] Y. Fathi. Computational complexity of LCP's associated with positive definite symmetric matrices. *Mathematical Programming*, 17:335–344, 1979.

[4] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[5] M. Kojima, S. Mizuno, and A. Yoshise. *A polynomial-time algorithm for a class of linear complementarity problems.* Research Reports on Information Sciences B-193, Dept. of Information Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro-ku, Tokyo 152, Japan, 1987.

[6] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming, Interior-Point and Related Methods*, pages 29–47, Springer-Verlag, New York, 1988.

[7] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

[8] K. G. Murty. Computational complexity of complementarity pivot methods. *Mathematical Programming Study*, 7:61–73, 1978.

[9] J. S. Pang, I. Kaneko, and W. P. Hallman. On the solution of some (parametric) linear complementarity problems with application to portfolio selection, structural engineering and actuarial graduation. *Mathematical Programming*, 16:325–347, 1978.

[10] L. Van der Hyden. A variable dimension algorithm for the linear complementarity problem. *Mathematical Programming*, 19:328–346, 1980.

[11] Y. Ye. *Further development on the interior algorithm for convex quadratic programming.* Integrated Systems Inc. and Department of Engineering-Economic Systems, Stanford University, Santa Clara, CA 95054 and Stanford, CA 94305, 1987.

Shinji MIZUNO: Department of Industrial Engineering and Manegement, Tokyo Institute of Technology
Oh-Okayama, Meguro, Tokyo 152, Japan