JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **45,** 91–103 (1997)
ARTICLE NO. PC971366

A Parallel Algorithm for Linear Programs with an Additional Reverse Convex Constraint[1]

Shih-Mim Liu* and G. P. Papavassilopoulos†

**Opto-Electronics and Systems Labs, Industrial Technology Research Institute, Chutung Hsinchu, Taiwan 310; and †Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, California 90089-2563*

A parallel method for globally minimizing a linear program with an additional reverse convex constraint is proposed which combines the outer approximation technique and the cutting plane method. Basically p ($\leq n$) processors are used for a problem with n variables and a globally optimal solution is found effectively in a finite number of steps. Computational results are presented for test problems with a number of variables up to 80 and 63 linear constraints (plus nonnegativity constraints). These results were obtained on a distributed-memory MIMD parallel computer, DELTA, by running both serial and parallel algorithms with double precision. Also, based on 40 randomly generated problems of the same size, with 16 variables and 32 linear constraints (plus $x \geq 0$), the numerical results from different number processors are reported, including the serial algorithm's.

Key Words: **reverse convex constraint; linear program; global optimization; parallel algorithm.**

1. INTRODUCTION

With rapidly advancing computer technology, particularly in the area of parallel machines, and the current advances in parallel algorithms (see, for example, [1, 2, 19, 20, 23, 24, 32]), solving nonconvex optimization problems for global optima using parallel algorithms seems to be considered computationally tractable. However, due to the variety of nonconvex problems and the absence of complete characterizations of global optimal solutions of nonconvex problems (e.g., there is no local criterion for deciding whether a local solution is global), it is necessary to devise parallel algorithms suited to particular classes of nonconvex problems. So far, although a large number of methods have been proposed, only a few of the presented algorithms have been programmed and tested. The aim of this paper is to introduce and study a parallel algorithm for a class of nonconvex problems and demonstrate its efficiency through extensive testing in a parallel machine (DELTA).

[1]Supported in part by the NSF under Grant CCR-9222734.

In the literature on nonconvex optimization problems, reverse convex programs, a problem closely related to concave minimization (cf. [4, 5, 11–14, 18, 25, 27, 29, 33]), has attracted the attention of a number of authors [6–9, 21, 22, 31] since Rosen [22] first studied it. The problem of linear programs with an additional reverse convex constraint is an interesting problem in reverse convex programs. Essentially, the feasible regions (i.e., intersection of a polyhedron and the complementary set of a convex set) for this class of optimization problems are nonconvex and often disconnected, and such feasible set results in the computational difficulty.

In recent studies for linear programs with one additional reverse convex constraint, Hillestad [7] developed a finite procedure for locating a global minimum. Hillestad and Jacobsen [8] gave characterizations of optimal solutions and provided a finite algorithm based on these optimality properties. Subsequently, Thuong and Tuy [28] proposed an algorithm involving a sequence of linear programming steps and concave programming steps. To increase efficiency, an outer approximation method in [13, p. 490] was used for the above concave programs. In addition, Pham Dinh and El Bernoussi [21] improved both the results and the algorithms described by Hillestad and Jacobsen [8], Thuong and Tuy [28]. For the procedure of Tuy cuts [29], Gurlitz and Jacobsen [6] showed that it ensures convergence for two-dimensional problems but not for higher-dimensional problems. They also modified the edge search procedure presented by Hillestad [7]. However, these are known to be rather time-consuming, or no computational experiments have been performed. Since the computational effort required strongly depends on the size of the problem and its type (e.g., linear objective function, linear constraints, or a reverse convex constraint), it is necessary to create an efficient algorithm to lower the computational load. A promising approach is to design a parallel algorithm for the above problems.

In this paper, we develop two new algorithms—serial and parallel algorithms—to solve linear programs with an additional reverse convex constraint. Basically, the serial algorithm can be regarded as a modification of algorithm 1 (first version) in Pham Dinh and El Bernoussi [21]. However,

0743-7315/97 $25.00

the serial algorithm presented here may be more efficient in many problems since it is based on the following: Theorem 4, cutting plane methods, and the outer approximation scheme with a simpler polyhedron S_k^0 (see Section 4). Generally speaking, the serial algorithm presented here has to solve both linear programs and concave minimization subproblems by the methods mentioned in [13, p. 490; 28]. The algorithm in [13, p. 490] seems to be more efficient than that in [28] because the latter requires more work to solve the concave programming subproblem, due to its lack of the outer approximation technique. In fact, for the outer approximation method, a solution of the concave minimization problem can frequently be found before revealing all the extreme points of the feasible set. The algorithm in [13, p. 490] constructs a decreasing sequence of S_k, i.e., $S_k \subset S_{k-1} \subset \cdots \subset S_0 \supset D$ (D is a bounded polyhedron; see Section 2), where $S_k = S_{k-1} \cup \{x \in \mathbb{R}^n: h(x) \leq 0\}$ or $S_k = S_{k-1} \cup \{x \in \mathbb{R}^n: cx \leq cx_k\}$ ($h(x)$ denotes a linear constraint of D, $x_k \in D$, and c is a cost vector). However, such a construction of the S_k may require us to do a rather expensive computation. For example, when $|V(S_k)|$, the number of vertices in S_k, is very large (this often occurs if n is large), the complexity of the computation of $V(S_{k+1})$ may increase considerably, in particular for forming $S_{k+1} = S_k \cup \{x \in \mathbb{R}^n: cx \leq cx_k\}$. Also, the storage of the vertices is a big problem. In this proposed serial algorithm, the reconstruction of a polyhedron S_k^0 (see Section 3) is proceeded as soon as a feasible point has been detected in step 2 of Phase II. With this construction of S_k^0, the amount of work required to calculate the newly generated vertices may be lower than that in [13, p. 490]; and the maximum memory to store the vertices can also be reduced. Although a simple construction of S_k^0 is used, the computation of generating new vertices is still the most expensive portion in the serial algorithm. To remedy this problem, a parallel algorithm was developed based on the serial algorithm. In Section 6, the computational results constitute a very important part of the present work since they employ a parallel machine (DELTA) to demonstrate that the parallel algorithm is accurate and efficient for the tested problems. For example, the parallel algorithm for 1, 16 processors and the serial algorithm have average computation times of 25.48, 3.47, and 75.16 s for 40 randomly created problems of the same size (32 constraints and 16 unknowns). Also, the different size tested problems with a number of variables up to 80 and 63 linear constraints can be solved in a good reasonable time.

The organization of this paper is as follows. In Section 2, the basic properties of optimal solutions for linear programs with an additional reverse convex constraint are stated. Section 3 is devoted to descriptions of the algorithms. Section 4 discusses the details of the implementation of our algorithms. In Section 5, two examples are presented to illustrate both serial and parallel algorithms. Finally, in Section 6, a numerical report including both serial and parallel algorithms running on the parallel machine DELTA is given.

2. PROBLEM STATEMENT AND BASIC PROPERTIES

This section introduces the main results corresponding to the characterization of optimal solutions of the linear program with an additional reverse convex constraint problem. Consider the problem

$$\text{(LRCP)} \quad \text{Minimize } \{cx: x \in D \cap G\}$$

where $D = \{x \in \mathbb{R}^n: Ax \leq b, x \geq 0\}$, A an $m \times n$ matrix, $b \in \mathbb{R}^m$, and $G = \{x \in \mathbb{R}^n: g(x) \geq 0\}$, g a finite convex function defined throughout $\mathbb{R}^n$. We use A_i to denote the ith row of A. Assume that D is bounded, and $D \cap G \neq \emptyset$. For any nonempty polyhedral set $D \subset \mathbb{R}^n$, we denoted by $V(D)$ the set of vertices of D, E(D) the set of edges of D, and ∂G the boundary of G for any nonempty set $G \subset \mathbb{R}^n$. Notice that, in general, a concave minimization problem

$$\text{(CP)} \quad \text{Minimize } \{f(x): \quad \text{s.t. } x \in D\}$$

where $f(x)$ is a continuous concave function on $\mathbb{R}^n$ and D is as in (LRCP), can be rewritten as a (LRCP) by introducing an additional variable t,

$$\text{Minimize } \{t: x \in D,\ \gamma_1 \leq t \leq \gamma_2,\ g(x, t) = t - f(x) \geq 0\}, \tag{1}$$

where γ_1, γ_2 are some constants in $\mathbb{R}$.

THEOREM 1 [8]. *Let D be a bounded polyhedron, and denote the convex hull of $D \cap G$ by* conv $(D \cap G)$. *Then we have:*

i. conv $(D \cap G)$ *is a bounded polyhedron and* conv $(D \cap G) =$ conv $(E(D) \cap G)$.
ii. *An optimal solution for* (LRCP) *lies in the set $E(D) \cap G$.*

THEOREM 2 [8]. *Let D be a bounded polyhedron, $y \in D \cap G$, and $D(y) = \{x \in D: cx \leq cy\}$. If for every $z \in V(D(y))$ we have*

i. $g(z) < 0$ *or*
ii. $g(z) = 0$ *and* $cz = cy$,

then y is an optimal solution for (LRCP).

DEFINITION 1 [13, 21]. The reverse convex constraint $G = \{x \in \mathbb{R}^n, g(x) \geq 0\}$ is called essential in the problem (LRCP) if we have

$$\min\{cx: x \in D\} < \min\{cx: x \in D \cap G\}.$$

COROLLARY 1 (see, e.g., [13]). *If the constraint $G = \{x \in \mathbb{R}^n: g(x) \geq 0\}$ is essential in* (LRCP) *and $D \cap G \neq \emptyset$ then there is an optimal solution for* (LRCP) *lying on $E(D) \cap \partial G$.*

Proof. Let $\overline{x}$ be an optimal solution to (LRCP). According to Theorem 1, $\overline{x}$ is a vertex of $\text{conv}(E(D) \cap G) = \text{conv}(E(D) \setminus \text{int}\, G^c)$ where $\text{int}\, G^c$ is the interior set of G^c, the complementary set of G. If $\overline{x} \notin \partial G$ then $g(\overline{x}) > 0$ and $\overline{x}$ must be a global solution to $\min\{cx: x \in D\}$. But G is essential in (LRCP). This implies that $g(\overline{x}) = 0$, i.e., $\overline{x} \in \partial G$. ■

Note that if the constraint G is not essential, then (LRCP) will be equivalent to the trivial linear programming problem, $\min\{cx: x \in D\}$. T. Pham Dinh and S. El Bernoussi [21] pointed out that if G is essential in (LRCP), then the sufficient condition in Theorem 2 is also necessary.

THEOREM 3 [21]. *Let D be a bounded polyhedron and let $x^* \in D$ be such that $g(x^*) = 0$. Let $D(x^*) = \{x \in D: cx \leq cx^*\}$. If the constraint G is essential in* (LRCP)*, then the necessary and sufficient condition for x^* to be an optimal solution of* (LRCP) *is that for each $v \in V(D(x^*))$ we have:*

i. $g(v) < 0$ or
ii. $g(v) = 0$ and $cv = cx^*$.

THEOREM 4. *Assume that $D \cap G \neq \emptyset$ and G is essential in* (LRCP). *Let $\overline{v} \in D$ and $D(\overline{v}) = \{x \in D: cx \leq c\overline{v}\}$. For a point $v^* \in D(\overline{v}) \cap G$ to be an optimal solution to the problem* (LRCP)*, a sufficient condition is*

$$\max\{g(x): x \in D(\overline{v})\} = 0 \text{ and}$$
$$cv^* = \min\{cx: g(x) = 0,\ x \in D(\overline{v})\}.$$

Proof. If $\max\{g(x): x \in D(\overline{v})\} = 0$ and $cv^* = \min\{cx: g(x) = 0,\ x \in D(\overline{v})\}$, then $g(x) \leq 0$ for each x in $D(\overline{v})$ and $cx \geq cv^*$ for each x in $D(\overline{v}) \cap G$. Hence, v^* is an optimal solution to (LRCP). ■

Remark. Let v^* be optimal; then the *sufficient condition* in Theorem 4 is also *necessary* if we have $\overline{v} = v^*$.

Note that $\max\{g(x): x \in D(\overline{v})\}$ is equivalent to $\min\{-g(x): x \in D(\overline{v})\}$, which is a concave minimization over $D(\overline{v})$. Hence, only the vertices of $D(\overline{v})$ are considered.

COROLLARY 2. *Assume that $D \cap G \neq \emptyset$ and G is essential. Let $\overline{v} \in D$ and $D(\overline{v}) = \{x \in D: cx \leq c\overline{v}\}$. For a point $v^* \in V(D(\overline{v})) \cap G$, if*

$$\max\{g(x): x \in V(D(\overline{v}))\} = 0 \text{ and}$$
$$c\overline{v} = \min\{cx: g(x) = 0,\ x \in V(D(\overline{v}))\},$$

then v^* is an optimal solution to problem (LRCP).

DEFINITION 2 [30]. Problem (LRCP) is said to be stable if

$$\min\{cx: x \in D,\ g(x) \geq \epsilon\} \downarrow$$
$$\min\{cx: x \in D,\ g(x) \geq 0\} \text{ as } \epsilon \downarrow 0.$$

If problem (LRCP) is stable, then one has $cv^* = c\overline{v}$ in Theorem 4 and it can be rewritten as following.

THEOREM 5 [30]. *Let $D \cap G \neq \emptyset$ and let G be essential. If the problem* (LRCP) *is stable, then a point $v^* \in D \cap G$ is an optimal solution to problem* (LRCP) *if and only if*

$$\max\{g(x): x \in D(v^*)\} = 0.$$

3. ALGORITHM DESCRIPTION

In this section, we present serial and parallel algorithms for (LRCP). Both algorithms are based on Theorems 1 and 4 and Corollary 1 and primarily make use of the outer approximation scheme and the cutting plane method.

3.1. Serial Algorithm

Initialization

Step 0. Let x_0 solve $\min\{cx: x \in D\}$ and let $v_0 \in V(D) \cap G$. Set $k = 0$.

Step 1. If $g(x_0) \geq 0$, stop; x_0 is optimal to (LRCP). Otherwise starting from x_0, pivot via the simplex algorithm for solving the linear programming $\max\{(v_0 - x_0)x: x \in D\}$ until a pair of vertices v_1 and v_2 are obtained such that $g(v_1) < 0$ and $g(v_2) \geq 0$.

Step 2. Solve the line search problem

$$\begin{aligned} &\text{Minimize} && \alpha \\ &\text{subject to} && g(v_1 + \alpha(v_2 - v_1)) \geq 0 \\ &&& 0 < \alpha \leq 1 \end{aligned}$$

and set $z_0 = v_1 + \overline{\alpha}(v_2 - v_1)$, where $\overline{\alpha}$ is an optimal value of the line search problem.

Step 3. Find a polyhedron S_0^0 containing a vertex x_0 such that $D(z_0) = \{x \in D: cx \leq cz_0\} \subset S_0^0 \subset \{x \in \mathbb{R}^n: cx \leq cz_0\}$, delete the redundant constraints according to $V(S_0^0)$, and go to Phase II.

Phase I

Step 1. Starting from x_k, pivot via the simplex algorithm for solving the linear programming $\min\{cx: x \in D(z_{k-1})\}$ until a pair of vertices v_1 and v_2 are obtained such that $g(v_1) \geq 0$ and $g(v_2) < 0$.

Step 2. Find $0 \leq \alpha < 1$ such that $g(v_1 + \alpha(v_2 - v_1)) = 0$ and set $z_k = v_1 + \alpha(v_2 - v_1)$, i.e., z_k is the intersection of $[v_1, v_2]$ with the surface $g(z_k) = 0$.

Step 3. Find a polyhedron S_k^0 of vertex x_0 such that $D(z_k) = \{x \in D: cx \leq cz_k\} \subset S_k^0 \subset \{x \in \mathbb{R}^n: cx \leq cz_k\}$ (such a polyhedron will be discussed later) and delete the redundant constraints according to $V(S_k^0)$.

Phase II

Let $i = 0$.

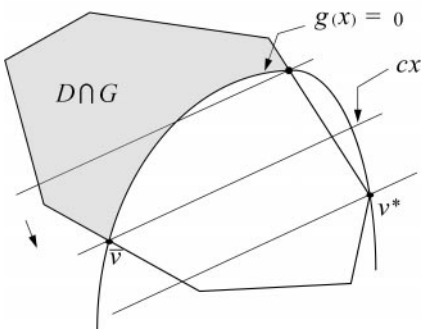

FIG. 1. An example for unstable problem.

Step 1. Set $\mathcal{V} = \{v\colon g(v) = 0, \text{ for } v \in V(S_k^i)\}$. If $\max\{g(v)\colon v \in V(S_k^i)\} = 0$ and $\mathcal{V} \subset D(z_k)$, then stop: $v^* \in \text{argMin}\{cx\colon g(x) = 0, x \in \mathcal{V}\}$ is a global solution for (LRCP). Otherwise, go to Step 2.

Step 2.

(a) If there is a $\overline{v} \in \text{argMin}\{cv\colon v \in V(S_k^i), g(v) \geq 0\}$ such that $\overline{v} \in D$, then set $x_{k+1} = \overline{v}$, $k = k + 1$, and go to Phase I. Otherwise, go to (b).

If there exists $\overline{v} \in \text{argMax}\{g(v)\colon v \in V(S_k^i), g(v) \geq 0\}$ such that $\overline{v} \notin D$, then find a constraint $p_j x - q_j \leq 0$ of D which is the most violated by $\overline{v}$ and set $S_k^{i+1} = \{x \in S_k^i\colon p_j x - q_j \leq 0\}$. Compute the vertex set V_k^{i+1} of S_k^{i+1} (from knowledge of V_k^i), and let $i = i + 1$, go to Step 1.

Remarks.

- For the search of any $v_0 \in V(D) \cap G$ in Step 0, see the methods described in Horst and Tuy [13] or Pham Dinh and El Bernoussi [21]. If no such v_0 exists, then there is no feasible solution.
- If x_0 is a degenerate vertex of D, then select another vertex x_0' such that x_0 is nondegenerate and $g(x_0') < 0$. If no such vertex exists, apply the procedures discussed in Section 4.
- Step 1 of Phase II is based on Theorem 4. For an unstable problem such as that in Fig. 1, Theorem 4 will be more efficient than Theorems 2 and 3.
- For a stable problem, step 1 of Phase II can be replaced by: If $\max\{g(v)\colon v \in V(S_k^i)\} = 0$, then stop: $v^* \in \text{argMin}\{cx\colon g(x) = 0, x \in V(S_k^i) \cap D(z_k)\}$ is a global solution for (LRCP). Otherwise, go to Step 2.

3.2. Parallel Algorithm

Initialization

Step 0. Let x^0 solve $\min\{cx\colon x \in D\}$ and $v^0 \in V(D) \cap G$. Set $k = 0$.

Step 1. If $g(x^0) \geq 0$, stop; x^0 is an optimal solution. Otherwise, execute the following in parallel: for processor i, find x^{0i}, a neighboring vertices of x^0 (if x^0 is a degenerate vertex, choose another vertex $\overline{x}^0$ where $g(\overline{x}^0) < 0$). Let $d^{0i} = v^0 - x^{0i}$.

If $g(x^{0i}) < 0$ then starting from x^{0i}, pivot via solving $\max\{d^{0i}x\colon x \in D\}$ until a pair of vertices v^{i1} and v^{i2} obtained such that $g(v^{i1}) < 0$ and $g(v^{i2}) \geq 0$.

Otherwise, set $v^{i1} = x^0$ and $v^{i2} = x^{0i}$.

Solve the line search problem with the serial algorithm and set $z_i^0 = v^{i1} + \overline{\alpha}(v^{i2} - v^{i1})$.

Step 2. Choose $z^0 = \min\{cz_i^0,\ i = 1, \ldots, n\}$ and do Step 3 of the initialization of the serial algorithm.

Phase I

Step 1. For point x^k and its n neighboring vertices x^{ki}, $i = 1, 2, \ldots, n$, processor i will do if $g(x^k) \geq 0$

- if $g(x^{ki}) < 0$ then set $v^{i1} = x^k$ and $v^{i2} = x^{ki}$. Otherwise, starting from x^{ki}, pivot via solving $\min\{cx\colon x \in D(z_{k-1})\}$ until a pair of vertices v^{i1} and v^{i2} obtained such that $g(v^{i1}) \geq 0$ and $g(v^{i2}) < 0$. Find z_i^k the intersection point of $[v^{i1}, v^{i2}]$ with the surface $g(z_i^k) = 0$.

Else

- if $g(x^{ki}) \geq 0$ then set $v^{i1} = x^k$ and $v^{i2} = x^{ki}$. Otherwise, set $d^i = v^0 - x^{ki}$, starting from x^{ki}, pivot via solving $\max\{d^i x\colon x \in D(z_{k-1})\}$ until a pair of vertices v^{i1} and v^{i2} obtained such that $g(v^{i1}) < 0$ and $g(v^{i2}) \geq 0$. Find z_i^k the intersection point of $[v^{i1}, v^{i2}]$ with the surface $g(z_i^k) = 0$.

Step 2. Find a polyhedron S_k^0 of vertex x^0 such that $D(z^k) = \{x \in D\colon cx \leq cz^k\} \subset S_k^0 \subset \{x \in \mathbb{R}^n\colon cx \leq cz^k\}$ and delete the redundant constraints according to $V(S_k^0)$, where $cz^k = \min\{cz_i^k\colon i = 1, 2, \ldots, n\}$

Phase II

Let $j = 0$.

Step 1. Set $\mathcal{V} = \{v\colon g(v) = 0, \text{ for } v \in V(S_k^i)\}$. If $\max\{g(v)\colon v \in V(S_k^i)\} = 0$ and $\mathcal{V} \subset D(z_k)$, then stop: $v^* \in \text{argMin}\{cx\colon g(x) = 0, x \in \mathcal{V}\}$ is globally optimal for (LRCP). Otherwise, go to Step 2.

Step 2.

(a) If there is a feasible vertex $\overline{v} \in \text{argMin}\{cv\colon v \in V(S_k^j), g(v) \geq 0\}$ such that $\overline{v} \in D$, then set $x^{k+1} = \overline{v}$, $k = k + 1$, and go to Phase I. Otherwise go to (b).

(b) If there exists $\overline{v} \in \text{argMax}\{g(v)\colon v \in V(S_k^j), g(v) \geq 0\}$ such that $\overline{v} \notin D$, then find a constraint $p_r x - q_r \leq 0$ of D which is the most violated by $\overline{v}$ and set $S_k^{j+1} = \{x \in S_k^j\colon p_r x - q_r \leq 0\}$. Compute the vertex set V_k^{j+1} of S_k^{j+1}(from knowledge of V_k^j), and let $j = j + 1$, go to Step 1.

Remarks.

- The line search in Step 1 of Initialization and Phase I can be performed $[n/p]$ times, where p is the number of processors and $[n/p]$ is the smallest integer which is greater than n/p.

• The (b) of Step 2 can be performed by a parallel computation. For example, let $V^+(S_k^j) = \{v \in V(S_k^j)\colon p_r v - q_r > 0\}$, where $p_r v - q_r = 0$ is a cutting plane. Denote by $|V^+(S_k^j)|$ the number of the vertices of $V^+(S_k^j)$. For a cutting plane $p_r v - q_r = 0$, $|V^+(S_k^j)| = M$, and $p =$ the number of processors, the calculation of the vertex set $V(S_k^{j+1})$ of S_k^{j+1} can be finished in $[M/p]$ times ($[M/p]$ is the smallest integer which is greater than M/p).

• (3) Similarly, the function values of $g(v)$, $v \in V(S_k^j)$, are computed in the same way as above.

LEMMA 1. *For Step* 1 *of Phase I, if both serial and parallel algorithms start from the same point* $\overline{x}$ *(i.e.,* $x_k = x^k$*), then* $cz^k(parallel) \le cz_k(serial)$*, i.e.,* $D(z^k) \subset D(z_k)$.

Proof.

(1) If $\overline{x}$ is a nondegenerate point, then we can find exactly its n adjacent vertices. Starting from these points, the edge search paths via the simplex algorithm will include the path in the serial algorithm. It implies $cz^k \le cz_k$.

(2) If $\overline{x}$ is a degenerate point, then we choose n neighboring vertices (n edge searching paths) including the path starting from $\overline{x}$.

From (1) and (2), we know that $cz^k \le cz_k$.

LEMMA 2. $\{cz_k\}$ *in the serial algorithm is a decreasing and finite sequence.*

See Hillestad and Jacobsen [8].

THEOREM 6. *The serial and parallel algorithms find an optimal solution for problem* (LRCP) *in a finite number of steps.*

Proof.

i. From Corollary 1, we know that there is an optimal solution for (LRCP) lying on $E(D)$. Now $E(D)$ is finite since the number of constraints on D is finite.

ii. At Step 3 of Phase I, both the polyhedron S_k^0 containing the global solution of (LRCP) and the number of linear constraints of $D(z^k)$, which is finite, imply that the number of cutting planes related to S_k^0 is finite.

Hence these two algorithms will converge to an optimal solution in a finite number of steps. ■

4. DISCUSSION OF IMPLEMENTATION

In the algorithms presented in Section 3, the outer approximation method was applied to solve the problem (LRCP), and there are two important procedures—edge searching and cutting plane procedures. Obviously, this approximation approach will be the most expensive computation in solving problem (LRCP) and its efficiency depends heavily on both construction of the polyhedron S_k^j and calculation of the vertex set $V(S_k^j)$. In other words, efficiency will increase if a suitable containing polyhedron is constructed. According to the techniques of outer approximation and of cutting plane the best choice of S_k^0 should be that it must be simple, be close enough to $D(z_k)$, and have a small number of vertices. Therefore, we would like to create the polyhedron S_k^0 mentioned in Step 3 of Phase I by using the same polyhedral cone (fixed constraints binding at x_0 or x^0). This is simpler than the construction described in Pham Dinh and El Bernoussi [21] since the latter has to find the n adjacent vertices and a linear variety generated by these n points, then solve a linear program.

Denote by $J(v)$ the index set of all constraints that are active at v ($v = x_0$ or x^0), i.e.,

$$J(v) = \{i \in I\colon p_i v - q_i = 0\}, \tag{2}$$

where $I \subset \mathbb{N}$ is a finite index set.

i. If v is a nondegenerate vertex of D, then $J(v)$ contains the indices of exactly n linearly independent constraints $p_i x - q_i = 0$, $i \in J(v)$. Let the set of inequalities

$$p_i x - q_i \le 0, \quad i \in J(v) \tag{3}$$

define a polyhedral cone vertexed at v. Therefore, the polyhedron (simplex) S_k^0 is defined as follows:

$$\begin{aligned} S_k^0 &= \{x \in \mathbb{R}^n\colon p_i x - q_i \le 0, \quad i \in J(v)\} \\ &\quad \cap \{x \in \mathbb{R}^n\colon cx \le cz_k \text{ or } cz^k\} \\ V(S_k^0) &= \{v, v_1, \ldots, v_n\}, \end{aligned} \tag{4}$$

where $v_1, \ldots, v_n$ can be obtained by the methods described in Pham Dinh and El Bernoussi [21]. Here the procedure of Horst *et al.* [26] was employed to generate them. Although S_k^0 is a simplex in most case, it could be unbounded. In an unbounded case, it may proceed in the following three ways:

• Try another v such that S_k^0 is bounded if there is a nondegenerate $v \in V(D)$ and $g(v) < 0$.

• Apply the methods mentioned in Pham Dinh and El Bernoussi [21] to construct a bounded approximation of S_k^0.

• Use an approximate cost vector c_ϵ instead of c. Since S_k^0 is unbounded, the number of vertices generated by the cutting hyperplane $cx - cz_k = 0$ will be less than n. Replace $s_j = 0$ by $s_j = \epsilon$ ($\epsilon > 0$) for some j in the procedure of Horst *et al.* [26] and do a pivot operation such that n vertices $v_1', \ldots, v_n'$ are generated. Hence one may have an approximate cutting plane $c_\epsilon x + \beta = 0$ which passes through these n points.

ii. If v is a degenerate vertex of D, then $|J(v)| > n$, i.e., there are more than n linear constraints binding at v and

$$V(S_k^0) = \{v, v_i;\ i = 1, 2, \ldots, q\} \quad \text{where } q > n. \tag{5}$$

In this case, one may apply the algorithm for finding all vertices of a given polytope (cf. [3, 15, 16]) or proceed with the methods mentioned in Pham Dinh and El Bernoussi [21]. Note that the algorithm of Matthess [15] needs to maintain a list structure; storage may thus be a problem for a large n.

Delete the redundant constraints by Horst *et al.*'s theorem [26] after each construction of S_k^0. For the determination of the vertex set $V(S_k^j)$ in the cutting plane procedures, the algorithms introduced above are sufficient. In addition, recall that global minimization of concave function (i.e., global maximization of convex function) always obtain its optimal solution at some vertex. Hence, to save storage memory and accelerate (b) of Step 2 in Phase II, only vertices with nonnegative convex function values need to be stored.

Let S_k^0 be a bounded polytope defined by the linear inequalities

$$S_k^0 = \{x \in \mathbb{R}^n\colon p_i x - q_i \le 0,\ i \in I\}. \tag{6}$$

Let $h(x) = p_j x - q_j = 0$, let $j \in I$ be a cutting hyperplane, and let $V_{g^+}(S_k^0) = \{v\colon g(v) \ge 0,\ v \in V(S_k^0)\}$. Let $V_{g^+}^+(S_k^0) = \{v \in V_g^+\colon h(v) > 0\}$. According to (b) of Step 2 in Phase II, a necessary condition for any constraint $h(x)$ of polyhedron S_k^0 to be a cutting plane is $V_{g^+}^+(S_k^0) \ne \emptyset$.

DEFINITION 3. A cutting plane $h(x)$ is valid for the polyhedron S_k^0 if $V_{g^+}^+(S_k^0) \ne \emptyset$. Otherwise, it is invalid.

Since only the vertex set $V_{g^+}(S_k^0)$ of $V(S_k^0)$, rather than $V(S_k^0)$, is useful for computation, a constraint which cannot be a cutting plane can be eliminated by the following lemma.

LEMMA 3. *Let $V_{g^+}(S_k^j) = \{v\colon g(v) \ge 0,\ v \in V(S_k^0)\}$. Then a constraint $h(x)$ of S_k^0 is invalid for S_k^0 if and only if*

$$h(v) < 0, \qquad \forall v \in V_{g^+}(S_k^0). \tag{7}$$

5. EXAMPLES

To illustrate the algorithms presented in Section 3, two examples are given here. In these two examples, a comparison of serial and parallel algorithms will be reported.

EXAMPLE 1.

Minimize:

$$-2x_1 + 3x_2$$

subject to:

$$\begin{aligned}
&-3x_1 + x_2 \le 0, \quad -4x_1 - x_2 \le -7,\\
&3x_1 + 2x_2 \le 23, \quad 5x_1 - 4x_2 \le 20,\\
&2x_1 + 3x_2 \le 22, \quad -6x_1 - 9x_2 \le -18,\\
&-3x_1 + x_2 \le 10, \quad x_1,\ x_2 \ge 0.\\
&g(x) = x_1^2 + x_2^2 - 8x_1 - 4x_2 + 13.75 \ge 0
\end{aligned}$$

SERIAL ALGORITHM.

Initialization. $x_0 = (4, 0)$ was obtained by solving the linear program $\min\{cx\colon x \in D\}$. Let $v_0 = (2, 6)$.

$k = 0$. Starting from x_0, Step 1 finds $v_1 = (5, 4)$, $v_2 = (2, 6)$ and Step 2 solves $z_0 = (4.2723, 4.4851)$. Step 3 constructs a simplex $S_0^0 = \{x \in \mathbb{R}^2\colon 5x_1 - 4x_2 \le 20,\ x_2 \ge 0,\ -2x_1 + 3x_2 \le 4.9107\}$ with vertices (4, 0), (−2.4554, 0), (11.3776, 9.2220) and deletes a redundant constraint $-3x_1 + x_2 \le 10$. Go to Phase II.

In Phase II, since $\max\{g(v)\colon v \in V(S_0^0)\} > 0$ and no $\overline{v}$ satisfies (a) of Step 2, set $S_0^1 = \{x \in S_0^0\colon 3x_1 + 2x_2 \le 23\}$. Thus, go to Step 1 of Phase II and obtain the same result as above. Form $S_0^2 = \{x \in S_0^1\colon -4x_1 - x_2 \le -7\}$ and find $x_1 = (1.1492, 2.4031) \in D$ in (a) of Step 1. Go to Phase I.

$k = 1$. In Phase I, Step 1 finds $v_1 = (1.5, 1)$, and $v_2 = (3, 0)$, Step 2 solves $z_1 = (1.8108, 0.7928)$, and Step 3 determines the simplex $S_1^0 = \{x \in \mathbb{R}^2\colon 5x_1 - 4x_2 \le 20,\ x_2 \ge 0,\ -2x_1 + 3x_2 \le -1.2431\}$ with vertices (4, 0), (0.6215, 0), (7.8611, 4.8264) and deletes a redundant constraint $-3x_1 + x_2 \le 0$. Similarly, executing the procedure of Phase II, we have $V(S_1^2) = \{(4, 0), (5.4989, 3.2516), (6, 2.5), (1.8108, 0.7928), (3, 0)\}$. Finally, Step 1 verifies that $v^* = z_1 = (1.8108, 0.7928)$ is an optimal solution because $\max\{g(v)\colon v \in V(S_1^2)\} = 0$ and only $g(z_1) = 0$.

PARALLEL ALGORITHM.

Initialization. In Step 0 we also find $x^0 = (4, 0)$ and let $v_0 = (2, 6)$.

$k = 0$. Step 1 finds $x^0 = (4, 0)$, $x^{01} = (6, 2.5)$, $x^{02} = (3, 0)$, $v^{11} = (5, 4)$, $v^{12} = (2, 6)$, $v^{21} = (3, 0)$, $v^{22} = (1.5, 1)$, and $z_1^0 = (4.2723, 4.4851)$, $z_2^0 = (1.8108, 0.7929)$. Hence $cz^0 = cz_2^0 = \min\{cz_1^0, cz_2^0\}$. Step 3 constructs a simplex S_0^0 in the same way as S_1^0 in the serial algorithm.

Phase II. After the same steps as the serial algorithm's, an optimal solution $v^* = z^0 = (1.8108, 0.7929)$ was discovered.

Figure 2 illustrates the geometric history of Example 1. In this example, the constraint $-3x_1 + x_2 \le 10$ is redundant.

EXAMPLE 2. Consider an example with $m = 10$ and $n = 6$. Cost vector: c = (–4.88166, –6.06580, –7.92004, 7.87233, –9.74772, –9.99590). Polyhedron: $D = \{x \in \mathbb{R}^n\colon Ax \le b,\ x \ge 0\}$, where $b^{\mathrm{T}} = (8.47832, 2.48636, 1.86858, -1.10545, 1.96469, 3.02506, 0.57517, -1.04776, 1.67729, 1.64407)$. Reverse convex constraint: $g(x) = 2.10(x_1 - 1.50)^2 + 5.24(x_2 - 3)^2 + 1.20x_3^2 + x_4^2 + 1.86(x_6 - 1)^2 - 45 \ge 0$.

$$A = \begin{pmatrix}
1.64990 & 1.78425 & 1.87958 & 0.13965 & 1.09823 & 1.92671\\
0.60304 & -0.73647 & 0.11778 & -0.40180 & 0.96786 & 0.83954\\
0.29427 & -0.22619 & 0.34955 & 0.81258 & -0.98052 & 0.46708\\
-0.37775 & -0.75322 & 0.56079 & -0.89150 & 0.59161 & -0.87597\\
-0.58633 & -0.52370 & 0.24838 & 0.51515 & 0.17007 & 0.32349\\
-0.01924 & 0.70454 & -0.74845 & 0.88219 & 0.90519 & -0.43513\\
-0.82323 & 0.02622 & 0.60894 & 0.52842 & -0.81546 & 0.58568\\
-0.50914 & 0.85069 & -0.49088 & -0.26491 & -0.40963 & -0.65797\\
0.88640 & -0.28695 & -0.81680 & 0.01408 & 0.69593 & 0.49501\\
-0.64534 & -0.19249 & 0.84724 & -0.37147 & 0.71062 & -0.55198
\end{pmatrix}.$$

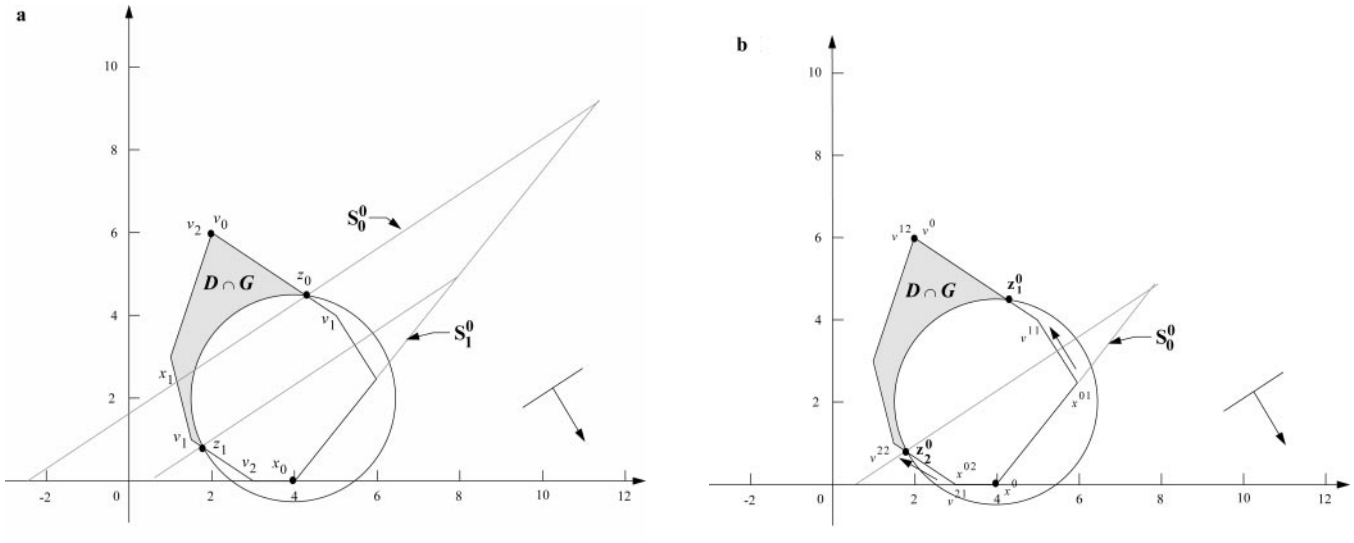

FIG. 2. Geometrical history of Example 1.

Choose $x_0 = x^0 = (0, 1.27570, 0.35670, 0, 1.98949, 1.73704)$ with $g(x_0) = -23.53236$ and $v_0 = v^0 = (1.10066, 0, 2.08640, 3.28416, 1.93021, 0.08425)$ with $g(v_0) = 20.06409$.

In the serial algorithm, there are 258 vertices generated by cuts and six constructed simplices having z_k $(k = 0, 1, \ldots, 5)$ as follows:

$$z_0 = (0.80574, 0.62003, 1.96709, 2.79354, 1.78034, 0), \quad cz_0 = -18.63633$$

$$z_1 = (0, 0.24375, 0.15993, 0, 0.48804, 1.48438), \quad cz_1 = -22.34023$$

$$z_2 = (0, 0.24350, 0, 0, 0.65845, 1.49730), \quad cz_2 = -22.86218$$

$$z_3 = (2.24338, 0.16203, 1.16530, 0, 0.23386, 1.05920), \quad cz_3 = -34.03075$$

$$z_4 = (0, 0.29620, 0.41783, 0, 1.03270, 1.97224), \quad cz_4 = -34.88672$$

$$z_5 = (1.19419, 0.17982, 1.36695, 0, 0.32943, 1.68998), \quad cz_5 = -37.85075.$$

After verification, we have a global optimal solution: z_5 with optimal value -37.85075. For the parallel algorithm with different numbers of processors, the results are listed in Table I. Note that only 24 new vertices and two simplices are generated before the optimal solution is found.

6. NUMERICAL RESULTS AND ANALYSIS

In this section, computational results are reported for solving problems (LRCP) by both the serial algorithm (SA) and the parallel algorithm (PA) described in Section 3 running on the DELTA supercomputer. The test problems are randomly generated so that the feasible region was nonempty and bounded.

6.1. DELTA and Test Problems

The Touchstone DELTA (cf. [17]) supercomputer is a message-passing multicomputer consisting of an ensemble of individual and autonomous nodes that communicate across a two-dimensional mesh interconnection network. It has 513 computational i860 nodes, each with 16 Mbytes of memory, and each node has a peak speed of 60 double-precision Mflops, 80 single-precision Mflops at 40 MHz. A concurrent file system (CFS) is attached to the nodes with a total of 95 Gbytes of formatted disk space. The operating system is Intel's Node Executive for the mesh (NX/M).

To share the information during the parallel computation, a node will be assigned as host node to collect the information

TABLE I
Iterative Results of Example 2 for PA with Processors 1, 2, 4, 6

Processors	$z^k (k = 0, 1, 2, 3)$	cz^k	Generated vertices
1	(3.20144, 0.41629, 1.28031, 0.33713, 0, 0)	−25.63958	114
	(1.19419, 0.17982, 1.36695, 0, 0.32943, 1.68998)	−37.85075	
2	(3.20144, 0.41629, 1.28031, 0.33713, 0, 0)	−25.63958	126
	(2.24338, 0.16203, 1.16530, 0, 0.23386, 1.05920)	−34.03075	
	(1.19419, 0.17982, 1.36695, 0, 0.32943, 1.68998)	−37.85075	
4	(0, 0.42927, 1.200501, 1.92978, 2.40445, 1.32133)	−33.56580	24
	(1.19419, 0.17982, 1.36695, 0, 0.32943, 1.68998)	−37.85075	
6	(0, 0.42927, 1.200501, 1.92978, 2.40445, 1.32133)	−33.56580	24
	(1.19419, 0.17982, 1.36695, 0, 0.32943, 1.68998)	−37.85075	

from each node and pass the messages to the others. The problems used to test the algorithms are randomly generated in the following way. The polyhedron $D = \{x \in \mathbb{R}^n: Ax \leq b, x \geq 0\}$ in a problem (LRCP), each element a_{ij} of A $(i = 1, \ldots, m; j = 1, \ldots, n)$ and b_i $(i = 1, \ldots, m)$, is obtained by (cf. Horst and Thoai [12])

$$a_{ij} = 2\theta_a - 1 \tag{8}$$

and

$$b_i = \sum_{j=1}^{n} a_{ij} + 2\theta_b, \tag{9}$$

where θ_a, θ_b are random numbers generated by the function RAND (a uniform distribution on [0, 1]) in MATLAB. Similarly, one can have c_j $(j = 1, \ldots, n)$, a coefficient of the linear cost function, in $[-10, 10]$ by $c_i = 10(2\theta_c - 1)$, where θ_c is also created by RAND.

Clearly, D is nonempty and bounded if we shift the elements of the first row of A by $a_{1j} = a_{1j} + 1$ such that all of its elements are positive.

For the reverse convex constraints, the following functions will be employed in the test problems,

$$\text{(I)} \quad g(x) = x^{\mathrm{T}} P x + rx - t,$$

$$\text{(II)} \quad g(u) = u^{\mathrm{T}} Q u + ru - t,$$

where P is a positive semidefinite $n \times n$ matrix, Q is a diagonal positive semidefinite $n \times n$ matrix, $r \in \mathbb{R}^n$, $t \in \mathbb{R}$, and a vector u consists of either x_i^2 (at least one) or x_i $(i = 1, \ldots, n)$.

$$\text{(III)} \quad g(x) = \left| x_1 + \frac{1}{2} x_2 + \frac{2}{3} x_3 + \cdots + \frac{n-1}{n} x_n \right|^{3/2} - t$$

$$\text{(IV)} \quad g(x) = \sqrt{1 + x_1 + 2x_2 + \cdots + nx_n} - t$$

Finally, solve $\min\{cx: x \in D\}$ and let $x_0 \in V(D)$ be its solution. Find a $v_0 \in V(D)$; then move the center of the convex functions near the x_0 so that $g(x_0) < 0$ and $g(v_0) \geq 0$.

6.2. Computational Results

Both serial and parallel algorithms were coded in standard Fortran 77. All numerical tests were performed on the parallel computer DELTA with double precision. In running the parallel algorithm for a test problem with n variables, p $(\leq n)$ nodes are used as a partition by specifying the numbers of rows and columns. Let PA(p) be the execution time for the parallel algorithm on p processors. Since PA(1) is not always less than SA, the speedup here is thereby defined as $\min(\text{SA}, \text{PA}(1))/\text{PA}(p)$. Tables II and III contain the computational results of the SA and PA described previously

TABLE II
Computational Results for Serial and Parallel Algorithms (I)

No.	Algorithm	N	m	n	RCC	Vmax	Vtol	Rec	Time (sec)	SpeedUp
1	Serial		27	9	(II)	151	1395	2	0.371	
	Parallel	1				154	1395	2	0.712	
		2				154	1395	2	0.445	0.83
		3				16	252	2	0.234	1.59
		4				16	252	2	0.219	1.69
		6				16	135	2	0.184	2.02
		9				16	135	2	0.180	2.06
2	Serial	1	63	9	(I)	92	10,125	4	2.496	
	Parallel	2				403	24,939	4	7.131	
		3				79	4896	2	1.698	1.47
		5				79	4896	2	1.308	1.91
		6				79	4896	2	0.749	3.33
		9				79	4896	2	0.924	2.70
						79	4896	2	0.740	3.37
3	Serial		40	10	(I)	91	8410	8	1.816	
	Parallel	1				60	4030	4	1.175	
		2				60	3880	4	0.856	1.37
		3				60	4030	4	0.794	1.48
		4				60	3880	4	0.670	1.75
		6				60	3880	4	0.638	1.84
		10				60	3880	4	0.513	2.29
4	Serial		36	12	(III)	688	10,092	4	3.257	
	Parallel	1				53	756	3	1.048	
		2				53	756	3	0.675	1.55
		3				53	756	3	0.571	1.84
		4				53	756	3	0.512	2.05
		6				53	756	3	0.471	2.23
		12				53	756	3	0.435	2.41

TABLE II—*Continued*

No.	Algorithm	*N*	*m*	*n*	RCC	Vmax	Vtol	Rec	Time (sec)	SpeedUp
5	Serial		30	16	(I)	5144	91,872	8	28.498	
	Parallel	1				7521	94,336	4	29.367	
		2				7521	89,360	3	15.773	1.81
		3				5144	41,488	4	5.756	4.95
		4				691	22,416	3	2.108	13.52
		8				691	26,096	3	1.559	18.28
		12				691	26,096	3	1.323	21.54
		16				691	26,096	3	1.161	24.55
6	Serial		42	22	(II)	12,654	688,512	10	369.737	
	Parallel	1				9708	248,468	5	126.503	
		3				8973	184,558	4	36.117	3.50
		4				8929	232,848	5	40.390	3.13
		9				8929	232,848	5	20.928	6.04
		12				8055	202,488	5	15.850	7.98
		16				8055	202,488	5	12.976	9.75
		22				8055	202,488	5	13.184	9.60
7	Serial		32	32	(I)	6168	166,624	12	122.941	
	Parallel	1				3960	28,096	4	27.077	
		2				7395	51,296	4	23.115	1.17
		4				7395	51,168	4	13.734	1.97
		9				1943	18,464	4	4.002	6.77
		16				2149	42,336	7	6.215	4.36
		20				1988	52,416	3	4.372	6.19
		25				1988	58,496	4	5.673	4.77
		32				1988	52,416	3	3.584	7.56
8	Serial		32	32	(I)	2241	54,944	10	40.393	
	Parallel	1				5603	44,192	4	36.231	
		4				5603	42,784	2	10.511	3.45
		8				5603	44,192	4	6.906	5.25
		9				1587	12,288	4	2.869	12.63
		16				1587	12,288	4	2.314	15.66
		25				503	5344	3	1.358	26.68
		32				503	5344	3	1.309	27.68

TABLE III
Computational Results for Serial and Parallel Algorithms (II)

No.	Algorithm	*N*	*m*	*n*	RCC	Vmax	Vtol	Rec	Time (sec)	SpeedUp
9	Serial		30	36	(I)	7121	93,564	15	73.226	
	Parallel	1				666	16,668	6	16.745	
		2				666	8424	4	6.242	2.68
		3				666	8424	4	4.327	3.87
		4				666	10,368	5	4.011	4.17
		8				100	9072	3	1.867	8.97
		9				225	14,508	5	2.678	6.25
		16				100	11,016	4	1.757	9.53
		25				100	9072	3	1.410	11.88
		36				100	9072	3	1.358	12.33
10	Serial		32	42	(I)	1265	52,248	7	49.568	
	Parallel	1				4387	47,376	5	56.763	
		2				807	22,302	4	14.502	3.42
		4				1265	15,792	2	5.807	8.54
		9				305	24,696	5	5.248	9.45
		16				305	24,696	5	4.204	11.79
		25				305	24,696	5	3.942	12.57
		35				305	18,522	4	3.143	15.77
		42				305	18,522	4	3.057	16.21

TABLE III—*Continued*

No.	Algorithm	*N*	*m*	*n*	RCC	Vmax	Vtol	Rec	Time (sec)	SpeedUp
11	Serial		30	45	(I)	2593	49,275	5	52.367	
	Parallel	1				2982	35,550	3	43.639	
		2				46	8640	4	7.297	5.98
		4				1031	34,290	3	11.408	3.83
		8				1031	33,660	3	6.135	7.11
		16				453	22,590	3	3.131	13.94
		25				46	7965	2	1.256	34.74
		36				46	7965	2	1.002	43.55
		45				46	7965	2	1.207	36.15
12	Serial		49	49	(IV)	9936	114,905	2	157.194	
	Parallel	1				48	2646	2	10.718	
		2				48	2646	2	6.008	1.78
		4				48	2646	2	3.430	3.12
		9				48	2646	2	2.204	4.86
		16				48	2646	2	1.701	6.30
		20				48	2646	2	1.487	7.21
		25				48	2646	2	1.805	5.94
		36				48	2646	2	2.078	5.16
		49				48	2646	2	1.543	6.95
13	Serial		27	55	(III)	9715	25,575	7	54.679	
	Parallel	1				6129	13,475	3	35.793	
		3				5923	12,760	3	13.111	2.73
		4				5923	12,760	3	10.412	3.44
		9				5923	12,760	3	7.102	5.04
		16				87	550	2	1.386	25.82
		25				87	550	2	1.260	28.41
		36				87	550	2	1.133	31.59
		49				87	550	2	1.594	22.45
		55				87	550	2	1.479	24.20
14	Serial		20	64	(I)	7095	67,264	15	145.011	
	Parallel	1				5011	25,536	7	70.949	
		2				2254	16,384	6	22.937	3.09
		4				2254	15,744	5	11.944	5.94
		8				2254	15,744	5	7.912	8.97
		16				2254	15,744	5	6.079	11.67
		30				1304	7360	3	3.622	19.59
		36				1304	13,824	5	5.096	13.92
		49				1304	13,376	4	5.807	12.22
		64				1304	13,376	4	5.571	12.74
15	Serial		30	80	(IV)	2067	87,440	9	223.437	
	Parallel	1				1765	72,080	6	207.123	
		2				3721	43,840	5	68.055	3.04
		5				2079	78,880	6	44.880	4.62
		10				1697	77,920	5	23.174	8.94
		16				1697	72,240	6	17.982	11.52
		20				1697	72,240	6	15.863	13.06
		25				1616	84,800	7	18.300	11.32
		30				2112	56,400	6	14.057	14.73
		35				68	320	3	2.743	75.51
		40				68	320	3	2.657	77.95
		49				72	320	3	2.808	73.76
		64				72	320	3	3.313	62.52
		80				72	320	3	3.802	54.48

on test problems of different sizes. Note that the choice of v^0 or v_0 may affect the time of calculation. However, so far, we have no general methods to choose it. In this paper, the point $\in V(D) \cap G$ in both SA and PA was the same (i.e., $v_0 = v^0$) for each tested problem. In order to demonstrate the efficiency of PA, we run 40 test problems randomly constituted with the same size ($m = 32$, $n = 16$). Also, a quadratic reverse convex constraint was considered for each problem. All numerical results are shown in Tables IV and V. Figure 3 shows the speedup (minimum, average, maximum) for 40 test problems

TABLE IV
Results for 40 Tested Problems with $m = 32$, $n = 16$ (I)

	Time(s)						
		Parallel (no. of processors)					
No.	Serial	1	2	3	4	9	16
1	20.481	9.853	5.500	5.134	4.216	2.197	1.826
2	2.238	1.775	1.454	0.799	1.006	0.363	0.417
3	78.932	40.368	17.089	12.177	4.229	2.597	1.958
4	15.155	7.208	2.391	3.220	2.988	1.757	1.056
5	23.248	1.939	1.364	1.166	0.990	0.943	0.805
6	2.931	3.706	2.205	1.577	1.242	0.244	0.201
7	106.328	82.587	27.962	23.750	16.497	20.771	8.894
8	35.369	1.511	0.946	0.762	0.588	0.530	0.522
9	29.709	33.203	8.847	10.277	5.384	4.982	4.037
10	15.239	9.330	5.077	3.806	3.059	2.054	1.608
11	3.293	1.242	1.020	0.704	0.572	0.576	0.605
12	53.568	28.821	15.365	11.006	8.641	2.963	2.349
13	17.022	4.842	2.497	1.745	1.402	0.773	0.755
14	33.843	7.927	4.962	2.985	3.523	1.195	1.079
15	33.060	19.448	13.823	7.565	5.104	2.466	1.650
16	5.356	2.844	1.256	0.895	1.015	0.571	0.536
17	7.526	1.301	0.839	0.565	0.460	0.365	0.367
18	35.761	7.807	4.313	3.687	2.734	0.875	0.521
19	16.129	8.377	2.445	1.633	1.315	0.379	0.405
20	58.590	9.756	4.396	3.218	2.593	0.778	0.588
21	444.697	21.016	14.579	7.785	8.687	3.976	2.946
22	585.474	100.211	56.529	106.228	47.860	25.322	24.219
23	644.556	238.583	130.017	96.096	82.259	52.525	40.964
24	39.629	45.967	19.831	14.592	10.401	6.282	5.978
25	24.315	9.519	5.345	3.776	2.709	1.891	1.248
26	11.687	2.874	1.629	1.319	1.034	0.648	0.603
27	18.878	14.459	7.705	1.200	4.907	0.651	0.558
28	99.030	29.773	15.699	13.024	8.747	6.015	3.497
29	2.263	3.012	0.527	0.350	0.271	0.182	0.204
30	40.990	4.800	2.862	2.246	1.797	1.012	0.707
31	31.163	18.103	8.304	8.312	5.569	3.822	2.831
32	10.655	3.668	1.245	0.906	0.779	0.572	0.648
33	27.489	2.892	1.817	1.153	1.132	0.676	0.735
34	33.292	3.198	1.855	1.423	1.151	0.804	0.826
35	43.266	33.983	18.651	13.890	11.368	7.982	6.705
36	43.704	30.517	18.943	4.297	9.144	3.750	2.206
37	202.924	132.465	90.057	15.368	59.096	12.227	8.464
38	37.961	23.100	11.336	8.564	6.875	1.175	1.094
39	26.450	6.789	3.914	2.900	2.287	1.726	1.338
40	44.337	10.374	6.718	4.462	4.078	3.003	2.812

TABLE V
Results for 40 Tested Problems with $m = 32$, $n = 16$ (II)

	Total number of generated vertices						
		Parallel (no. of processors)					
No.	Serial	1	2	3	4	9	16
1	82,944	35,968	35,968	46,080	48,480	41,184	41,360
2	7728	4288	5472	4288	5792	1072	1072
3	296,128	144,704	111,280	110,064	50,960	50,960	50,960
4	44,368	19,168	10,368	21,296	24,144	19,616	10,368
5	75,680	3072	3072	3072	3072	2944	2944
6	8848	8848	8848	8848	8848	64	64
7	372,752	280,736	181,616	192,640	181,616	228,880	228,880
8	113,392	2496	2528	2528	2528	2528	2528
9	101,360	103,264	47,568	81,488	47,568	75,376	75,376
10	56,768	30,432	30,432	30,432	30,432	30,432	30,432
11	9616	2176	2544	2064	2176	2672	2624
12	161,600	87,152	87,152	87,152	88,640	46,320	46,320
13	50,736	10,432	10,208	10,208	10,208	10,000	10,272
14	101,456	21,616	26,320	21,616	34,336	16,976	21,616
15	133,648	73,344	96,704	73,344	62,048	49,600	39,280
16	15,888	5440	3408	3408	5392	2352	2720
17	23,248	1456	1456	1456	1456	848	848
18	92,432	24,976	24,432	28,608	25,376	6720	4336
19	44,656	23,616	5664	5664	5664	928	928
20	150,624	28,272	23,136	22,080	23,136	6944	6144
21	1,000,112	72,048	96,592	67,616	98,816	67,616	63,968
22	1,285,776	278,080	278,080	638,128	304,288	355,088	445,616
23	1,707,776	682,592	653,616	653,616	684,880	758,768	752,672
24	119,808	130,752	98,064	98,064	98,064	86,544	117,328
25	88,704	36,656	36,928	36,656	31,200	31,200	29,488
26	40,464	7824	7824	7824	5568	5568	7632
27	61,552	43,280	42,656	48,816	7984	7984	7984
28	302,288	91,120	89,456	89,456	109,168	82,944	109,168
29	7360	7360	720	720	304	304	304
30	114,608	11,696	12,192	12,480	10,016	9216	12,496
31	108,480	67,312	56,368	79,472	67,760	74,736	74,752
32	41,328	10,432	4592	3840	4592	3840	3840
33	83,504	7840	8272	6224	8272	4784	4784
34	96,416	8560	8560	8480	8560	6688	6688
35	111,120	88,336	88,336	88,336	88,513	96,688	101,440
36	135,472	92,112	105,616	35,168	89,408	72,192	56,064
37	580,560	433,952	473,168	144,464	495,360	257,648	228,816
38	105,408	66,928	55,328	55,328	55,968	14,608	17,040
39	76,928	17,568	18,416	18,416	18,416	19,920	19,920
40	102,384	21,360	25,648	23,104	26,272	28,896	33,616

of the same size ($m = 32$, $n = 16$). The average computation times of PA for 1, 16 processors and SA are 25.479, 3.469, and 75.163 s, respectively. Tables II, III, IV, and V illustrate that the PA introduced here is very efficient for the solution of the tested problems.

In our computational experiment, the computational load of SA and PA depends on the type of (LRCP) problem, determined by its cost function, its linear constraints, and a reverse convex constraint. A different cost function will produce a different sequence of $\{S_k^0\}$, more linear constraints may cause more cuts, and the reverse convex constraint is related to the locations of z_k (or z^k). In general, for the same tested problem, the set $\{z_k\}$ in SA is not necessary to contain the set $\{z^k\}$ in PA and $|z_k|$ is frequently greater than $|z^k|$, where $|z_k|$ ($|z^k|$) is the number of $\{z_k\}$ ($\{z^k\}$) (see Examples 1, 2, or Rec in Tables II, III). Also, it may vary for the set $\{z^k\}$ in PA for different numbers of processors (Tables II, III). Compared with SA, PA will frequently decrease the number of cuts during the compu-

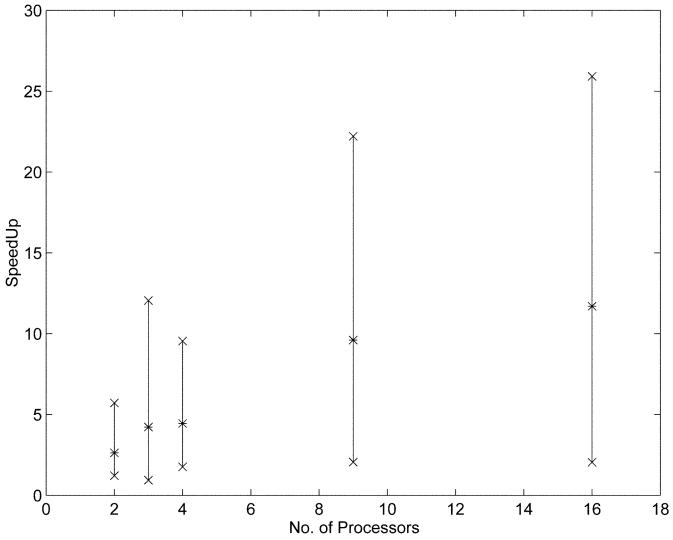

FIG. 3. Speedup for 40 problems with $m = 32$, $n = 16$ on the DELTA.

tation resulting in a lower number of newly generated vertices, even for PA with single processor. In addition, the vertices created by cuts can be computed in parallel for PA. Thus, PA is much more efficient than SA. Moreover, observing the variation of the set $\{z^k\}$ in PA, speedups greater than the number of processors can be expected in some test problems (Tables II, III, IV, V). Notice that with the same total number of generated vertices, PA with single processor is slower than SA because the former has to find the n adjacent vertices and do pivoting and edge searching for each adjacent vertex (cf. 1 in Table II and 6, 29 in Tables IV, V). However, Tables II, III, IV, V show that the number of new vertices produced by cuts for PA with single processor is frequently much lower than that created in SA. Therefore, it is expected that the efficiency of SA can be much improved in most test problems if SA takes additional time to execute the edge searching procedure as PA.

The notations employed in Tables II, III are as follows:

N:	number of processors
m:	number of constraints in $Ax \leq b$
n:	number of variables
RCC:	type of Reverse Convex Constraint described in Section 6.1
Rec:	number of polyhedra S_k^0 constructed, $k = 0, 1, 2, \ldots$
Vmax:	maximal number of vertices stored
Vtotal:	total number of vertices generated by cuts (not including $V(S_k^0),\ k = 0, 1, 2, \ldots$).

The parallel algorithm introduced in this paper is a synchronous parallel procedure since the subsequent step will not be executed until completing the computation of the previous step. For example, in Phase I, if one wants to do Step 2, one has to finish Step 1 and obtain z^k, the minimum of $z_i^k,\ i = 1, \ldots, n$. Here, various numbers of processors $p\ (\leq n)$ are used to solve the (LRCP) problem for parallel algorithm. From the computational results, we know that a high efficiency may be achieved if a suitable number of processors are chosen. In fact, in some problems, using more processors may not be realistic because many processors may be idle during the computation and more processors cause more communication overhead. Finally, since the memory required to store the list of vertex increases rapidly with n, the size of problem is restricted. Although CFS (Concurrent File System) can be used for the larger size of problem, it requires an inordinately long time to complete read/write processes.

7. CONCLUSION

In this paper, a new parallel algorithm has been proposed to solve the problem (LRCP) that can be efficiently implemented on a massive parallel computer DELTA. We have tested two sets of randomly generated test problems. For the first set, we emphasized problems of different sizes; for the other set, we concentrated on problems of the same size ($m = 32$, $n = 16$).

In the algorithm presented here, the calculation of producing new vertices is the most expensive part. However, this computation is distributed over all processors and saves a considerable amount of time although it requires the communication. By comparing it with the serial algorithm, we have achieved computational results (Tables II, III, IV, V) that show the parallel algorithm for different numbers of processors is more efficient, with even a superlinear speedup for some tested problems. As mentioned in the preceding section, greater than linear speedup is caused by different choices in the search process, but there is no method to predict them. The numerical experiments show that the PA for 1-processor case seems to have better performance than the SA in most tested problems.

REFERENCES

1. Bertsekas, D. P., and Tsitsiklis, J. N. *Parallel and Distributed Computation: Numerical Methods.* Prentice–Hall, Englewood Cliffs, NJ, 1989.
2. Bixby, R. E., Kilgore, A., and Torczom, V. Very large-scale linear programming: A case study in exploiting both parallelism and distributed memory. *6th SIAM Conference on Parallel Processing for Scientific Computing.* Norfolk, VA, 1993.
3. Dyer, M. E., and Proll, L. G. An algorithm for determining all extreme points of a convex polyhedron. *Math. Programming* **12** (1977), 81–91.
4. Falk, J. E., and Hoffman, K. L. A successive underestimation method for concave minimization problems. *Math. Oper. Res.* **1** (1975), 251–259.
5. Falk, J. E., and Hoffman, K. L. Concave minimization via collapsing polytopes. *Oper. Res.* **34** (1986), 919–929.
6. Gurlitz, T. R., and Jacobsen, S. E. On the use of cuts in reverse convex programs. *J. Optim. Theory Appl.* **68** (1991), 257–274.
7. Hillestad, R. J. Optimization problems subject to a budget constraint with economies of scale. *Oper. Res.* **23** (1975), 1091–1098.
8. Hillestad, R. J., and Jacobsen, S. E. Linear programs with additional reverse convex constraint. *Appl. Math. Optim.* **6** (1980), 257–269.
9. Hillestad, R. J., and Jacobsen, S. E. Reverse convex programming. *Appl. Math. Optim.* **6** (1980), 63–78.

10. Ho, H. F., Chen, G. H., Lin, S. H., and Sheu, J. P. Solving linear programming on fixed-size hypercubes. *Proceedings of International Conference on Parallel Processing.* 1988, pp. 112–116.
11. Hoffman, K. L. A method for globally minimizing concave functions over convex sets. *Math. Programming* **20** (1981), 22–32.
12. Horst, R., and Thoai, N. V. Modification, implementation and comparison of three algorithms for globally solving linearly constrained concave minimization problems. *Computing* **42** (1989), 271–289.
13. Horst, R., and Tuy, H. *Global Optimization.* Springer-Verlag, Berlin, 1990.
14. Liu, S. M., and Papavassilopoulos, G. P. A parallel method for globally minimizing concave functions over a convex polyhedron. *2nd IEEE Mediterranean Symposium on New Directions in Control and Automation.* 1994.
15. Mattheiss, T. H. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Oper. Res.* **21** (1973), 247–260.
16. Mattheiss, T. H., and Rubin, D. S. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Math. Oper. Res.* **5** (1980), 167–185.
17. Messina, P. The concurrent supercomputing consortium: Year 1. *IEEE Parallel Distrib. Technol.* (1993), 9–16.
18. Pardalos, P. M., and Rosen, J. B. Methods for global concave minimization: A bibliographic survey. *SIAM Rev.* **28** (1986), 367–379.
19. Pardalos, P. M. (Ed.). Advances in Optimization and Parallel Computing, Honorary Volume on the Occasion of J. B. Rosen's 70th Birthday. North-Holland, Amsterdam, 1992.
20. Phillips, A. T., Pardalos, P. M., and Rosen, J. B. *Topics in Parallel Computing in Mathematical Programming.* Science Press, 1993.
21. Pham Dinh, T., and El Bernoussi, S. Numerical methods for solving a class of global nonconvex optimization problems. In *New Methods in Optimization and Their Industrial Uses.* Birkhäuser Verlag, Basel, 1989, pp. 97–132.
22. Rosen, J. B. Iterative solution of nonlinear optimal control problems. *SIAM J. Control* **4** (1966), 223–244.
23. Rosen, J. B., and Maier, R. S. Parallel solution of large-scale, block-diagonal linear programs on a hypercube machine. *Proceedings of the 4th Conference on Hypercube Concurrent Computers and Applications.* 1989, pp. 1215–1218.
24. Stunkel, C. B., and Reed, D. C. Hypercube implementation of the simplex algorithm. *Proceedings of the 4th Conference on Hypercube Concurrent Computers and Applications.* 1989.
25. Thieu, T. V. Improvement and implementation of some algorithms for nonconvex optimization problems. Lecture Notes in Mathematics, Vol. 1405, pp. 159–170. Springer-Verlag, Berlin, 1989.
26. Thoai, N. V., Horst, R., and de Vries, J. On finding new vertices and redundant constraints in cutting plane algorithms for global optimization. *Oper. Res. Lett.* **7** (1988), 85–90.
27. Thoai, N. V., Horst, R., and Tuy, H. Outer approximation by polyhedral convex sets. *Oper. Res. Spektrum* **9/3** (1987), 153–159.
28. Thuong, T. V., and Tuy, H. A finite algorithm for solving linear programs with an additional reverse convex constraint. Lecture Notes in Economics and Mathematical Systems, Vol. 225, pp. 291–302. Springer-Verlag, Berlin, 1984.
29. Tuy, H. Concave programming under linear constraints. *Soviet Math. Dokl.* **4** (1964), 1437–1440.
30. Tuy, H. A general deterministic approach to global optimization via D.C. programming. *FERMAT Days 85: Mathematics for Optimization.* North-Holland, Amsterdam, 1986, pp. 273–303.
31. Tuy, H. Convex programs with an additional reverse convex constraint. *J. Optim. Theory Appl.* **52** (1987), 463–486.
32. Wu, M., and Li, Y. Fast LU decomposition for sparse simplex method. *6th SIAM Conference on Parallel Processing for Scientific Computing.* Mar. 1993.
33. Zwart, P. B. Nonlinear programming: Counterexample to two global optimization algorithms. *Oper. Res.* **21** (1973), 1260–1266.

Received December 20, 1994; revised July 14, 1997; accepted July 15, 1997

SHIH-MIM LIU received his B.S. in engineering science from the National Cheng Kung University, Taiwan, in 1984; his M.S. in electrical engineering from the New Jersey Institute of Technology, Newark, NJ, in 1989; and his Ph.D. in electrical engineering from the University of Southern California in 1995. He joined Opto-Electronics and Systems Laboratories, Industrial Technology Research Institute, Taiwan, as a research staff member in 1995. His research interests include parallel algorithms for nonconvex optimization problems, control systems, color printer systems, and color spectral measurement systems.

G. P. PAPAVASSILOPOULOS obtained his Diploma in electrical and mechanical engineering from the National Technical University of Athens, Greece, in 1975, and his M.S. (1977) and Ph.D. (1979) in electrical engineering, both from the University of Illinois at Urbana-Champaign. He is currently a professor of electrical engineering at the University of Southern California. His research interests are control, optimization, and game theory.