JUNE 19 - 22, 1994
MALEME-CHANIA, CRETE, GREECE

A Parallel Method for Globally Minimizing Concave Functions Over a Convex Polyhedron *

Shih-Mim Liu and G. P. Papavassilopoulos
Dept of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2563

Abstract. An algorithm for globally minimizing concave functions over a bounded polyhedron is described. A crucial feature of the algorithm is that it can be implemented in parallel, thus enhancing the rapidity of convergence. The algorithm generates a sequence of nested polyhedra containing the polyhedron of the constraint set. The initial containing polyhedron with N vertices is obtained by solving some linear programming problems. Beginning with the vertices of the initial polyhedron, the algorithm proceeds simultaneously with N subproblems whose solutions minimize the concave function over N successively tighter polyhedra created by the cutting-plane method. The algorithm is guaranteed to converge to the global solution. Computational considerations of the algorithm are discussed.

Key words: Simplex, Cutting Plane, Parallel Algorithm, Globally Minimizing Concave Function.

I. Introduction

In recent years, a rapidly growing number of papers has been published pertaining to solving specific classes of multiextremal global optimization problems (cf.,e.g.,[3, 8]). Several of these papers are on the topic of globally minimizing a concave function over a polyhedron which Tuy[10] first addressed in 1964. Most approaches that have been developed for minimizing a concave function over a polyhedron have been based mainly on the following approaches: cutting planes, successive approximation, successive partition, or combinations of these methods. Only a few of the suggested algorithms have been demonstrated by numerical tests, including the two successive approximation methods elaborated by Falk and Hoffman[5, 6] and the three different efficient algorithms discussed by Horst and Thoai[12].

In this paper, we consider the problem of minimizing a concave function subject to a linear polyhedron constraint set. In our method, an initial enclosing polyhedron with N vertices (e.g. a simplex) has to be computed by finding some vertices of the feasible set. Then, by suitable partition of this enclosing polyhedron , we have N independent subproblems with the same objective function. After initialization, an upper bound on the objective function is obtained, and the algorithm proceeds to solve these N subproblems in parallel. The comparison of the upper bounds of the concave function for all subproblems will be considered during the period of computation, and the algorithm is guaranteed to converge to the global solution. The algorithm we present here uses a combination of cutting planes, outer approximation, and partition techniques with the method of cutting planes playing a dominant role. The algorithm must generate the new vertices in defining the new polyhedron by cut and this is its most expensive computation. Several algorithms for finding all vertices of a polyhedron are compared in [4], and the method developed by Mattheiss[1] seems to be the most efficient. In our algorithm, we will incorporate the idea proposed by Host, Thoai and Vries[2] and the method of Mattheiss[1] in the calculation of all new generated vertices.

The algorithm introduced in this paper, the methods proposed by Falk and Hoffman[5], and the algorithm OAA described by Horst and Thoai[12] have similar characteristics, i.e. successively outer approximation and finite procedures. However, the choices of the cutting plane are different in these three methods, in particular, our parallel algorithm is much more efficient than the others. The comparison of the computations in some methods[3, 5, 12] will be given in Section V.. Generally speaking, our method has the following properties: it can find accurately all global optimal solutions in finite iterations, it is a parallel algorithm, it can handle the degenerate case (which is not considered in [5, 6]) (see Host, Thoai and Vries[2]), the redundant constraints do not affect the computation because they will never be chosen as a cutting plane during the computation, and it does not require a separable objective function.

In the next Section will present the details of the algorithm. In Section 3, we introduce a scheme. In Section 4, we give two examples to illustrate the method. Section 5 reports some computational results. Finally, in the Appendix we discuss the extension of the method to globally minimizing a concave function over a compact convex feasible set, and a small illustrative example is provided.

*supported in part by NSF under Grant CCR-9222734

II. Problem Statement and Algorithm

Consider the problem

$$(\mathcal{P}) \quad \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & Ax \le b, \quad x \ge 0. \end{cases}$$

where A is an m×n matrix, $b \in R^m$, $x \in R^n$, and f is a concave function defined throughout R^n. We use a_i and b_i to denote the ith row of A and b respectively. Assume D=$\{x \mid Ax \le b,\ x \ge 0\}$ is bounded, nonempty, and there is a point p in the strict interior of the feasible region D. i.e. p$\in \{x : a_i x < b_i,\ x > 0 \text{ for } i = 1, \ldots, m.\}$.
The following theorem is well known:

Theorem 1 *If a global minimum of a concave function over any polyhedron is attained, it can always attained at some vertex of the polyhedron.*

A. Serial Algorithm

In the algorithm, we can choose an initial enclosing simplex D^0 by solving n+1 linear programming problems of the form:

$$\begin{cases} \min x_j & (j = 1, 2, \ldots, n) \\ \text{s.t. } x \in \text{D} \end{cases} \quad \begin{cases} \max \sum_{i=1}^{n} x_i \\ \text{s.t. } x \in \text{D} \end{cases} \quad (1)$$

Let $\alpha_j (j = 1, 2, \ldots, n)$ and α be the optimal value solutions of (1) occurring at the points p_j $(j = 1, 2, \ldots, n+1)$ respectively which are vertices of D. Then it is easily seen that

$$D^0 = \{x \in R^n : x_j \ge \alpha_j\ (j = 1, \ldots, n), \sum_{j=1}^{n} x_j \le \alpha\} \quad (2)$$

is a simplex containing D. Denote by $V(D)$ the vertex set of any polyhedron D. Obviously, the vertex set of D^0, $V(D^0) = \{v_1, v_2, \ldots, v_{n+1}\}$, where $v_{n+1} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $v_j = (\alpha_1, \alpha_2, \ldots, \alpha_{j-1}, \beta_j, \alpha_{j+1}, \ldots, \alpha_n)$ $(j = 1, 2, \ldots, n)$ with $\beta_j = \alpha - \sum_{i \ne j}^{n} \alpha_i$. For n=2, An enclosing simplex D^0 is shown in Figure 1. Clearly an upper bound

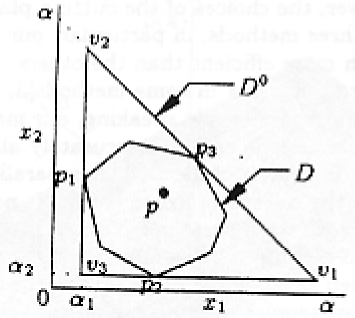

Figure 1: A Simplex containing D for n=2

for $(\mathcal{P})$ is $f^0 = \min\{f(p_i),\ i = 1, 2, \ldots, n+1\}$ since the p_i are vertices of D. Let vertex v^0 be a solution of the problem $\min\{f(v_j) :\ v_j \in V(D^0)\}$. Then $f(v^0) \le f^0$ since $D^0 \supseteq D$(We only need to store vertices v_j, where $f(v_j) \le f^0$, $v_j \in V(D^0)$). If $v^0 \in D$, then v^0 solves problem $(\mathcal{P})$. If $v^0 \notin D$, we find a linear constraint from $Ax = b$, say $a_i x = b_i$, such that $a_i z = b_i$ and $z \in$ boundary of D, where $z = v^0 + \lambda(p - v^0)$, $0 < \lambda < 1$(Recall that p is an interior point of D, and thus $\lambda \ne 1$. Notice also that $\lambda \ne 0$ since $v^0 \notin D$). Thus, we get a new polyhedron $D^1 = D^0 \bigcap \{x \in R^n :\ a_i x \le b_i\}$, and $V(D^1)$. Following the same procedure as before with D^1 assuming the role of D^0, the algorithm will generate a sequence of subproblems, and there will be a corresponding sequence of polyhedra D^k.

Let I^k be the subset of $\{1, 2, \ldots, \text{m}\}$(i.e. of the constraints set $\{a_i x \le b_i,\ i = 1, 2, \ldots, m\}$) whose corresponding constraints are not used in defining D^k. We can thus state the following algorithm:

Initialization:

Take an n-simplex $D^0 \supset D$ by solving (1), and find f^0, and $V(D^0)$. Only the v_i's for which $f(v_i) \le f^0$, for $v_i \in V(D^0)$, need be stored. Set I^0 = the constraint set of D = $\{1, 2, \ldots, \text{m}\}$.

Iteration $k = 1, 2, \ldots$

At iteration k, we know V^{k-1}, the promising vertex set of D^{k-1} where $f(v) \le f^{k-1}$, *for* $\forall v \in V^{k-1}$.
Choose x^k by solving $\min\{f(v) :\ v \in V^{k-1}\}$(if more than one solution, choose any one). Compute

$$(\mathcal{L}^k) \begin{cases} \max\ \lambda \\ \text{s.t.: } x^k + \lambda(p - x^k) \in \{x : a_j x = b_j,\ j \in I^{k-1}\}, \\ 0 \le \lambda < 1. \end{cases}$$

If $\lambda = 0$(if there is no λ for problem $(\mathcal{L}^k)$, set $\lambda = 0$), then x^k is a global minimum solution of $(\mathcal{P})$, and $f^* = f(x^k)$. Otherwise, set $z^k = x^k + \lambda(p - x^k)$, and choose any one constraint such that $a_j z^k - b_j = 0$, $j \in I^{k-1}$. Form

$$D^k = D^{k-1} \bigcap \{x \in R^n : a_j x \le b_j\}, I^k = I^{k-1} \backslash \{j\} \quad (3)$$

Compute the new vertices generated by the cutting plane $a_j x - b_j = 0$ (only store the vertices yielding objective function values $\le f^k$).
If $V^k = \phi$, then stop. Otherwise, go to iteration $k+1$.

B. Parallel Algorithm

In order to cast the previously described algorithm in a parallel form, we assume that we have n+1 processors, and an initial enclosing simplex D^0 was obtained by solving (1) on n+1 processors. For the ith processor $(i = 1, \ldots, n)$, set

$$D_i^0 = \{x \in R^n : \sum_{j=1}^{n} x_j \le \alpha,\ \alpha_i < x_i$$
$$\alpha_j \le x_j (j = 1, \ldots, i-1, i+1, \ldots, n)\} \quad (4)$$

and for the $(n+1)$th processor, set

$$D_{n+1}^0 = \{x \in R^n : \alpha_j \le x_j\ (j = 1, \ldots, n), \sum_{j=1}^{n} \alpha_j < \alpha\} \quad (5)$$

Then,

$$D \subset \bigcup_{i=1}^{n+1} D_i^0 = D^0, \text{ and } V(D_i^0) = v_i \ (i = 1, \ldots, n+1) \quad (6)$$

So, problem $(\mathcal{P})$ can be rewritten as follows:

$$(\mathcal{P}_i) \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in D_i^0 \bigcap D, \ i = 1, 2, \ldots, n+1. \end{cases}$$

where the solution of $(\mathcal{P})$ = min {the solutions of $(\mathcal{P}_i)$ $\ldots, (\mathcal{P}_{n+1})$}. There are n+1 independent subproblems which are similar to the serial algorithm stated in Section A. Thus, these subproblems could be done in parallel. Denote S^k by the set of D_i^k's (i.e. the number of the active processors at iteration $k \geq 1$)= $S^{k-1} \setminus N^{k-1}$, where $N^{k-1} = \{i : f(v) > f^{k-1}$, for every $v \in V(D_i^{k-1})$, $i \in (1, 2, \ldots, n+1)\}$, and let J_i^k be the subset of {1,2,...,m} whose corresponding constraints were not used in defining D_i^k, $i \in S^{k-1}$. Let V_i^k denote the set of the promising vertices of D_i^k. The algorithm is then as follows:

Initialization:

Construct $D_i^0 (i = 1, \ldots, n+1)$ and n+1 independent subproblems described as (4), (5) and $(\mathcal{P}_i)$ respectively.

Set f^0 = min $\{f(p_i), \ i = 1, \ldots, n+1\}$ (the best upper bound so far). We keep the $D_i^{0'}s$ with $i \in S^0$ and go to iteration 1.

Iteration k =1,2,...

At this iteration, set $S^k = S^{k-1} \setminus N^{k-1}$; D_i^{k-1}, V_i^{k-1} the promising vertex set of the D_i^{k-1} and a current upper bound f^{k-1} are known for every $i \in S^{k-1}$. Then, select the most promising vertex of the D_i^{k-1} by choosing min { $f(v) : v \in V_i^{k-1}$ }, and let x_i^k be the solution(if more than one solution, take one of them). Solve

$$(\mathcal{L}_i^k) \begin{cases} \max \lambda \\ \text{s.t.:} x_i^k + \lambda(p - x_i^k) \in \{x : a_j x = b_j, j \in J_i^{k-1}\}, \\ \quad 0 \leq \lambda < 1. \end{cases}$$

If $\lambda = 0$ (if there is no λ for problem $(\mathcal{L}_i^k)$, set $\lambda = 0$), then x_i^k is a vertex of D, and let $f_i^k = f(x_i^k)$ if $f(x_i^k) < f^{k-1}$ or $f_i^k = f_i^{k-1}$ if $f(x_i^k) = f^{k-1}$ (keep x_i^k, it may be one of the global optimal solutions).

If $\lambda > 0$, then set $z_i^k = x_i^k + \lambda(p - x_i^k)$, find all constraints of D which are binding at z_i^k and set $I_i^k = \{j : a_j z_j^k = b_j, \ \ j \in J_i^{k-1}\}$. Then add any one constraint $a_j x - b_j \leq 0$, $j \in I_i^k$ to generate D_i^k by using the cutting plane method. Set

$$D_i^k = D_i^{k-1} \bigcap \{x : a_j x \leq b_j, \ \ j \in I_i^k\}, \ J_i^k = J_i^{k-1} \setminus I_i^k \quad (7)$$

$$\begin{aligned} V_i^k &= \{\text{the promising vertices of the subset } D_i^k\} \\ &= \{v : f(v) \leq f_i^k, \text{ for } v \in V(D_i^k)\} \end{aligned} \quad (8)$$

Now update the upper bound by

$$f^k = \min \{f^{k-1}, f_i^k \text{ (if } f_i^k \text{ available)}; \quad i \in S^{k-1}\}. \quad (9)$$

Set $N^k = \{i : \ \ min(f(v) : v \in V_i^k) > f^k$ or $V_i^k = \phi$, for $i \in S^{k-1}\}$

If $S^k = \phi$, then stop the algorithm, and f^k is the global minimum of problem $(\mathcal{P})$; otherwise set $k = k+1$, and go to iteration k.

Lemma 2 *The sequence of upper bounds* $\{f^k\}$ *is monotonically decreasing.*

Proof: $f^k = \min\{f(x), \ \ \forall x \in V(D^k) \bigcap V(D)\}$, where $V(D^k) = \bigcup_{i=1}^{n+1} V(D_i^k)$, and min $\{f(x) : x \in V(D^{k+1}) \bigcap V(D)\} \leq$ min $\{f(x) : x \in V(D^k) \bigcap V(D)\}$ since $D \subseteq D^{k+1} \subseteq D^k \subseteq \ldots \subseteq D^0$. Thus $f^{k+1} \leq f^k$. ☐

Lemma 3 *The algorithm will terminate in a finite number of iterations.*

Proof: Because of D is a polyhedron, it has a finite number of vertices , and at most m cuts will be introduced for determining D_i^0. Notice also that the number of new points generated by the cutting plane method is finite. It follows that the algorithm must terminate in a finite number of iterations. ☐

Theorem 4 *The algorithm converges, and* f^k *is the global minimum of problem* $(\mathcal{P})$ *if it terminates at iteration* k.

Proof: By lemma 3, assume the algorithm terminates at the k-th iteration. Thus $f^k \leq f^{k-1} \leq \ldots \leq f^1 \leq f^0$ (by lemma 2), but f^k =min$\{f^{k-1}, f_i^k$; *for* $i \in S^{k-1}\} \Rightarrow f^k$ is the global minimum of problem $(\mathcal{P})$. ☐

III. A Scheme With $n^2 + 1$ Processors

In the parallel algorithm, an initial enclosing polyhedron must be computed first. Since the performance of the parallel procedure depends heavily on the kind of the initial containing polyhedron, other alternatives could be considered. For instance, we can have an n-rectangular enclosing polyhedron (with 2^n vertices) by solving 2n linear programming problems of the form:

$$\begin{cases} \min x_j \ (j = 1, \ldots, n) \\ \text{s.t. } x \in \text{D} \end{cases} \quad \begin{cases} \max x_j \ (j = 1, \ldots, n) \\ \text{s.t. } x \in \text{D} \end{cases} \quad (10)$$

However, so far, there are no general rules for the construction of the initial enclosing polyhedron. Now, we are going to introduce an enclosing polyhedron Q^0 for which the maximal number of active processors can be up to $n^2 + 1$ at the first iteration (in general, the number of the active processor is less than $n^2 + 1$).

First, we compute the following linear programming problems on 2n+1 processors.

$$\alpha_j = \{\min x_j, \text{ s.t. } x \in \text{D}\}, \ (j = 1, \ldots, n) \quad (11)$$

$$\alpha = \{\max \sum_{i=1}^{n} x_i, \text{ s.t. } x \in \text{D}\} \quad (12)$$

$$\beta_j = \{\max x_j, \text{ s.t. } x \in \text{D}\}, \ (j = 1, \ldots, n) \quad (13)$$

Let

$$Q^0 = D^0 \bigcap \{x \in R^n : x_i \leq \beta_i, i = 1, \ldots, n\} \quad (14)$$

$$Q_i^0 = D_i^0 \bigcap \{x \in R^n : x_i \le \beta_i\}, i = 1, \ldots, n \quad (15)$$

$$Q_{n+1}^0 = D_{n+1}^0 \bigcap \{x \in R^n : x_i < \beta_i, i = 1, \ldots, n\} \quad (16)$$

Clearly, there are n vertices in $V(Q_i^0)$ the vertex set of Q_i^0. Set $v_{i_j} \in V(Q_{i_j}^0)$ and

$$\begin{aligned} Q_{i_j}^0 \quad = \quad & \{x \in R^n : x_l > \alpha_l, (l = 1, \ldots, n), \sum_{l=1}^n x_l < \alpha\} \\ & \bigcap \{x \in R^n : x_i \le \beta_i\} \bigcup \\ & \{\text{all constraints of } Q_i^0 \text{ binding at } v_{i_j}\}, \quad (17) \end{aligned}$$

where $i, j = 1, 2, \ldots, n$. Therefore,

$$D \subset \bigcup_{i=1}^{n+1} Q_i^0 = \bigcup_{i=1}^{n} \bigcup_{j=1}^{n} Q_{i_j}^0 \bigcup Q_{n+1}^0 \quad (18)$$

Figure 2 gives the illustration of these subsets for n=2.

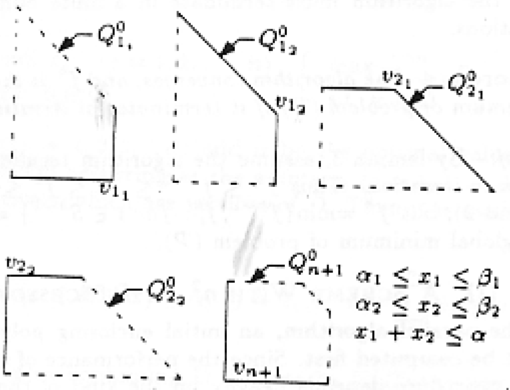

Figure 2: $Q_{i_j}^0$ for n=2

Equation (18) implies that problem $(\mathcal{P})$ can be represented by

$$(\mathcal{Q}_{i_j}) \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in Q_{i_j}^0 \bigcap D, \quad i, j = 1, 2, \ldots, n \end{cases}$$

and

$$(\mathcal{Q}_{n+1}) \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & x \in Q_{n+1}^0 \bigcap D \end{cases}$$

where the solution of $(\mathcal{P})$ = min {the solutions of $(\mathcal{Q}_{i_j})(i, j = 1, \ldots, n)$, and $(\mathcal{Q}_{n+1})$}. Hence, at most $n^2 + 1$ subproblems will be solved simultaneously by a procedure similar to the one stated in Section B.

IV. Example

In order to illustrate the method, we present two examples here. In example 1, we will introduce both serial and parallel algorithms. Only the parallel method will be applied to example 2.

Example 1:

$$\begin{aligned} \text{minimize} \quad & \frac{-((x_1-1)^2 - 2x_1x_2 + (x_2-2)^2)}{2x_1} \\ \text{subject to:} \quad & -3x_1 + x_2 \le 0, && \zeta_1 \\ & -4x_1 - x_2 \le -7, && \zeta_2 \\ & 3x_1 + 2x_2 \le 23, && \zeta_3 \\ & 5x_1 - 4x_2 \le 20, && \zeta_4 \\ & 2x_1 + 3x_2 \le 22, && \zeta_5 \\ & -6x_1 - 9x_2 \le -18, && \zeta_6 \\ & -15x_1 + 5x_2 \le 10, && \zeta_7 \\ & x_1, x_2 \ge 0. \end{aligned}$$

serial algorithm: the initial enclosing simplex D^0 was

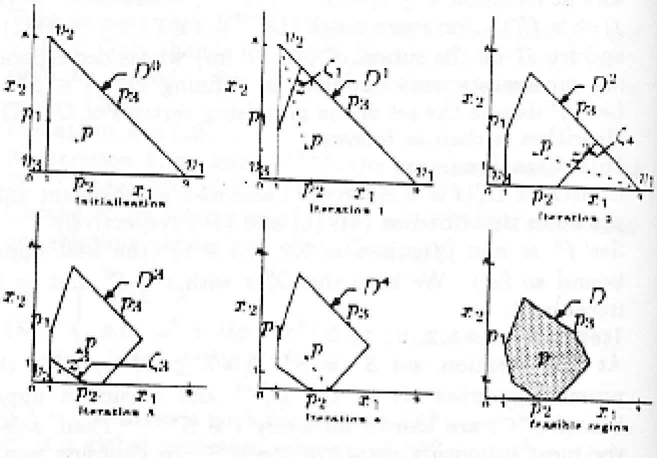

Figure 3: Serial algorithm for example 1

obtained by generating the vertices $v_1 = (9, 0), v_2 = (1, 8), v_3 = (1, 0)$, and $p_1 = (1, 3), p_2 = (3, 0), p_3 = (5, 4)$ are the vertices of D yielding function values of $2.50, -1.333$, and 2.0 respectively. Thus $f^0 = -1.333$, $N^0 = \phi$ since $f(v_1) = -3.778, f(v_2) = -10.0, f(v_3) = -2.0 < f^0$, i.e. $S^0 = \{1,2,3\}$. Let $I^0 = \{1, 2, \ldots, 7\}$, and interior point $p = (2.600, 1.867)$.
Iteration 1: obviously, $x^1 = v_2$. To solve $(\mathcal{L}^1)$, we get $\lambda = 0.457$ and $z^1 = (1.732, 5.195)$. Therefore, the cutting plane is ζ_1, and the new vertices are (2.25,6.75),(1,3) having function values of 1.389,2.5 respectively. So, no new points will be kept, and $I^1 = \{2, 3, \ldots, 7\}$, $V^1 = \{(9,0), (1,0)\}$, $f^1 = -1.333$.
Iteration 2: $x^2 = v_1$, thus $z^2 = (4.946, 1.182)$ with $\lambda = 0.633$. It follows that the cutting plane is ζ_4, and the new vertices are (4,0), (6.222,2.778). The objective function values are $-1.625, 0.538$ respectively. Hence, (4,0) need to be stored. $I^2 = \{2, 3, 5, 6, 7\}$, $V^2 = \{(1,0), (4,0)\}$. The upper bound does not improve, i.e. $f^2 = -1.333$.
Iteration 3: choose $x^3 = v_3$, then $z^3 = (1.732, 0.849)$ with $\lambda = 0.455$. Therefore, the cutting plane is ζ_3. The new vertices and corresponding function values are $f(3, 0) = f(p_2)$, and $f(1, 1.333) = 1.111$. Then, we have $I^3 = \{2,5,6,7\}$, $V^3 = \{(4,0), (3,0)\}$. The upper bound is still the same.
Iteration 4: since $x^4 = (4, 0)$, we get $\lambda = 0$. (4,0) is feasible and $f^4 = f(4, 0) = -1.625$. It implies $V^4 = \phi$. The algorithm terminates here, and (4,0) is the global solution.

Table 1: Iterative Results for Example 2

k		Processors (i) 1	2	3	4	S^k	f^k
0	$f(p_i)$	f(0,3.5,1) = −6.000	f(2.727,0,2.182) = −5.267	f(1,1.5,0) = −4.000	f(1.463,1.854,2.098) = −0.349	{1,2,3,4}	−6.000
	$f(v_i)$	f(5.415,0,0) = −25.739	f(0,5.415,0) = −20.324	f(0,0,5.415) = −14.910	f(0,0,0) = −7.250		
1	λ	0.8951	0.8010	0.4745	0	{1,2,3,4}	−7.250
	z_i^1	(1.497,1.227,0.945)	(0.832,2.176,0.846)	(0.493,0.650,3.346)	(0,0,0)		
	Cutting Plane	ζ_1	ζ_1	ζ_3			
	New Vertices	(2.000,0,0) (-4.829,10.244,0) (2.854,0,2.561)	(-4.829,10.244,0) (0,3.000,0) (0,3.805,1.610)	(4.244,0,1.171) (0,2.829,2.585) (0,0,4.000)			
	The Promising Vertices	(2.000,0,0)	(0,3.000,0)	(4.244,0,1.171)			
	Lower Bound	−7.250	−7.250	−13.461	−7.250		
2	λ	0.4220	0	0.8225		{1,2,3}	−7.250
	z_i^2	(1.594,0.578,0.446)	(0,3.000,0)	(1.607,1.127,1.076)			
	Cutting Plane	ζ_2		ζ_1			
	New Vertices	(1.000,1.500,0) (2.333,0,1.000) (1.000,0,0)		(1.463,1.854,2.098) (2.727,0,2.182) (2.854,0,2.561)			
	The promising Vertices			(0,0,4)			
	Lower Bound	−6.250	−7.250	−7.250			
3	λ			0		(3)	−7.250
	z_i^3			(0,0,4.000)			
	Lower Bound			−7.250			

Figure 3 illustrates the sequence of containing polyhedra generated by the serial algorithm. In this example, constraint ζ_7 is redundant.

parallel algorithm: we need 3 processors to solve this

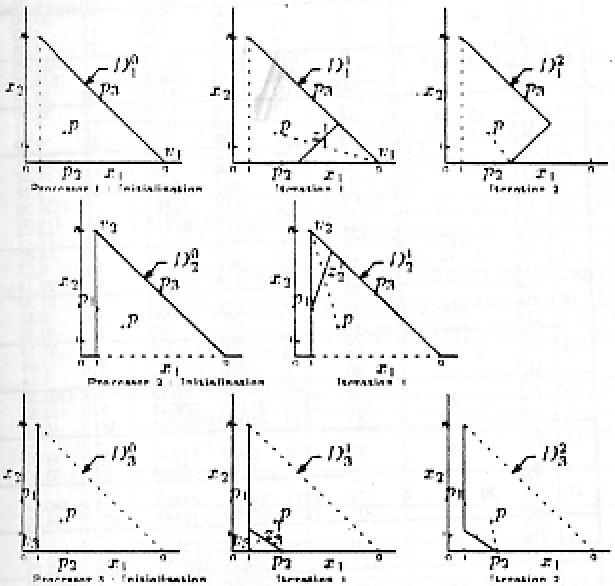

Figure 4: Parallel algorithm for example 1

problem. After initialization, we proceed as follows:

Iteration 1:

Processor 1: choose $x_1^1 = v_1$. Solving $(\mathcal{L}_1^1)$, we have z_1^1 =(4.946,1.182), and λ =0.633. Therefore, the cutting plane is ζ_4. The new vertices and corresponding function values are $f(4,0) = -1.625$ and $f(6.222,2.778)$=0.538. So, the lower bound is -1.625, I_1^1 ={4}, J_1^1 ={1,2,3,5,6,7}, V_1^1 = {(4,0)}. Processor 2: take $x_2^1 = v_2$. Then, z_2^1 =(1.732,5.195) with λ =0.457. Thus, the cutting plane is ζ_1, and the new vertices and their objective function values are f(2.25,6.75)=1.389 and f(1,3)=2.50. Hence, $I_2^1 =\{1\}$, $J_2^1 = \{2,\ldots,7\}$, $V_2^1 = \phi$. Processor 2 stops.

Processor 3: for $x_3^1 = v_3$, we get z_3^1 =(1.727,0.849) with λ =0.455. Therefore, the cutting plane is ζ_3, and the new vertices are (3,0), (1,1.333) yielding function values of −1.333, 1.111 respectively. I_3^1 = {3}, J_3^1 ={1,2,4,5,6,7}, V_3^1 ={(3,0)}. At the end of iteration 1, the upper bound does not improve, i.e. $f^1 = f^0 = -1.333$, N^1={2}, S^1 ={1,3}.

Iteration 2:

Processor 1: with λ =0, x_1^2 =(4,0) is feasible. Then, $f_1^2 = f(4,0) = -1.625$, $V_1^2 = \phi$, and processor 1 stops.

Processor 3: similarly, we have $f_3^2 = f(3,0) = -1.333$, $V_3^2 = \phi$, stop.

Now, since N^2 ={1,3}, $S^2 = \phi$, the algorithm stop, and $f^2 = f_1^2 = -1.625$ is the global minimum solution. The history of these processors is shown in Figure 4.

Example 2 was constructed so that it has three global minimizers, and a redundant constraint ζ_5 is included in the constraint set.

Example 2:

$$\begin{array}{lll} \text{minimize} & -(x_1-1)^2-(x_2-1.5)^2-(x_3-2)^2 & \\ \text{subject to:} & 3x_1+2x_2-x_3 \le 6, & \zeta_1 \\ & 3x_1-4x_3 \le 3, & \zeta_2 \\ & 4x_1+3x_2+6x_3 \le 24, & \zeta_3 \\ & 2.25x_1+4x_2+3x_3 \le 17, & \zeta_4 \\ & x_1+2x_2+x_3 \le 10, & \zeta_5 \\ & x_1,x_2,x_3 \ge 0. & \end{array}$$

After the initialization (inter point p = (1.04, 1.37, 1.06)), the algorithm required three iterations to find all the global solutions (0,0,0), (0,3,0), and (0,0,4) with the same function value of −7.250. All the calculations can be found in Table 1.

V. Computational Results

In this section, we present several computational experiments. Computational results for similar problems to those considered here are also presented in [5] and [12]. Our algorithm seems to perform very well in comparison to the results reported in [5, 12], basically due to its parallel character. The largest problem solved in [5] is a 12-variable, 19-constraint problem and the algorithm OAA in [12] was run only for problems with $n \leq 12$. But, using our parallel algorithm, we can efficiently solve similar test problems of different size with up to 50 variables and 30 linear constraints (aside from nonnegativity constraints).

Here we first compare the results using our algorithm on the examples in [3, 5, 12] as following:

(1) example in [5], we have $D^0 = \{x \in R^3 : x_1 \geq 0.7286,\ x_2, x_3 \geq 0,\ \sum_{i=1}^{3} x_i \leq 3.1333\}$, $p = (0.8981, 0.0800, 0.7743)$, global solution=(1,0,0) with value -1. SA: Iter=11, Vmax=7, Vgen=12, Vtotal=26, Time=2.2848 sec; PA: Iter=4, Vmax=6, Vgen=10, Vtotal=26, $N_a = (4,4,4,2,1)$, Time=0.3990 sec.

(2) example on page 249 or 306 in [3], we have $D^0 = \{x \in R^4 : x \geq 0,\ \sum_{i=1}^{4} x_i \leq 5.0755\}$, $p = (0.5, 0.5, 0.15, 2.0)$, global solution = (0.7904,0.4824,0,3.8027) with value -7.6558, redundant constraint: $1.2x_1 + 1.4x_2 + 0.4x_3 + 0.8x_4 \leq 6.8$. SA: Iter=4, Vmax=3, Vgen=8, Vtotal=20, Time=1.9761 sec; PA: Iter=4, Vmax=2, Vgen=8, Vtotal=20, $N_a = (5,2,1,1,1)$, Time=0.5932 sec.

(3) example in [12], we have $D^0 = \{x \in R^5 : x_1 \geq 0.0715,\ x_2, x_4 \geq 0,\ x_3 \geq 0.2820,\ x_5 \geq 0.3606, \sum_{i=1}^{5} x_i \leq 12.5720\}$, $p = (1.5, 2.0, 3.0, 0.7, 1.0)$, global solution =(0.4096,5.6011,6.1354,0, 0.4258) with value -21.1220, redundant constraint: $0.7620x_1 - 0.3048x_2 - 0.0123x_3 - 0.3940x_4 - 0.7921x_5 \leq 1.2057$. SA: Iter=7, Vmax=10, Vgen=45, Vtotal=120, Time=4.6049; PA: Iter=6, Vmax=8, Vgen=35, Vtotal=130, $N_a = (6,4,4,3,3,2,1)$, Time=1.1898 sec.

All computational results including Table 2 are obtained from MATLAB 4.0 running on Sun 4/50 (SPARCstation IPX) with 16 MB of memory. The test problems for Table 2 are randomly constructed by hand. Here, we use a definition of speedup which is the ratio between the time taken by a given serial computer executing the serial algorithm and time taken by the parallel computer(imitated by the same serial computer) executing the parallel algorithm using N processors. In order to imitate the parallel algorithm by serial computer, we assume that all active processors have the same computing time in the same iteration. Then, the CPU time executed by parallel algorithm will be $Time = \sum_{i=0}^{k} T_a(i)/N_a(i)$; where $T_a(i)$ is the time taken by all active processors running at i-th iteration, N_a is the number of active processors at i-th iteration. In our computational experiments, the performance of PA is pretty good in general, and QA is often a very efficient algorithm for problem ($\mathcal{P}$) when A is symmetric. Table 2 shows the results for the test problems of different size which have the same concave objective function of the form

$$f(x) = -\frac{1}{n} \sum_{i=1}^{n} x_i^2 - 1 \qquad (19)$$

The interior point p of each test problem is the same for algorithms SA, PA, and QA. In our computational experience, the performance of SA, PA, and QA heavily depend on the choice of the interior point p. However, we have no general rules for this choice.

Table 2: Computational Results for SA, PA, QA

No	Algo	m	n	Iter	N	Vmax	Vgen	Vtotal	Pmax	Time	Sno
1	SA	5	3	[illegible]	[illegible]	2	4	23	1	1.47	1
	PA			3	8	3	7	17	4	0.33	
	QA			2	5	1	8	10	[illegible]	0.26	
2	SA	6	4	6	6	4	12	32	1	2.26	1
	PA			4	12	4	16	36	5	0.46	
	QA			4	25	3	47	71	12	0.45	
3	SA	6	5	11	11	16	75	170	1	6.09	1
	PA			10	46	40	175	455	6	2.56	
	QA			3	143	30	300	920	20	1.55	
4	SA	21	5	6	6	28	45	124	1	8.54	1
	PA			6	24	29	63	181	6	2.02	
	QA			3	24	11	60	100	13	1.03	
5	SA	7	6	5	8	6	25	109	1	5.48	1
	PA			7	32	16	84	260	7	1.85	
	QA			8	102	67	285	755	30	1.65	
6	SA	8	6	4	4	7	42	76	1	4.82	1
	PA			5	24	11	48	140	7	1.18	
	QA			5	29	12	89	188	15	1.26	
7	SA	6	7	10	10	23	161	361	1	16.11	1
	PA			7	41	32	136	466	8	3.44	
	QA			9	184	87	479	1330	42	2.73	
8	SA	6	8	10	10	33	210	570	1	25.92	1
	PA			9	49	48	220	737	9	9.15	
	QA			8	229	124	593	2133	56	5.15	
9	SA	6	9	6	6	10	137	235	1	20.18	2
	PA			6	21	7	72	149	10	1.91	
	QA			9	38	32	154	348	19	3.72	
10	SA	10	9	411	411	401	619	3121	1	297.13	81
	PA			171	1131	1041	2795	12074	10	151.42	
	QA			87	3068	4750	24888	69003	72	135.24	
11	SA	6	10	6	6	30	235	581	1	29.59	2
	PA			6	29	22	168	411	11	2.64	
	QA			9	145	71	806	2031	90	5.97	
12	SA	10	10	4	4	77	498	626	1	37.70	1
	PA			3	26	73	320	580	11	4.02	
	QA			2	19	9	88	88	21	1.08	
13	SA	10	10	24	24	1857	3213	10274	1	1461.80	10
	PA			11	110	4066	7578	21679	11	544.70	
	QA			7	360	200	1300	4172	100	33.10	
14	SA	15	12	5	5	73	788	1499	1	98.40	1
	PA			4	48	81	708	1612	13	7.41	
	QA			11	39	47	456	827	25	9.29	
15	SA	19	12	6	6	164	1716	2922	1	196.57	1
	PA			6	52	173	1440	2380	13	14.38	
	QA			13	131	99	857	2000	55	9.28	
16	SA	20	12	9	9	724	4170	15481	1	1536.90	1
	PA			10	111	999	4585	22318	13	119.96	
	QA			9	544	697	6277	24311	133	27.77	
17	SA	15	45	6	6	10	161	340	1	54.07	1
	PA			6	43	23	262	796	16	6.57	
	QA			13	57	66	613	1050	31	26.13	
18	SA	15	15	4	4	16	761	399	1	57.30	1
	PA			6	48	46	459	1060	16	9.82	
	QA			4	29	14	185	199	31	10.38	
19	SA	5	20	22	22	27	378	378	1	52.18	1
	PA			4	57	77	531	1469	21	7.09	
	QA			1	20	0	0	0	41	1.14	
20	SA	12	20	[illegible]	[illegible]	946	3964	26733	1	4393.30	1
	PA			7	107	2112	21176	39643	21	162.83	
	QA			5	781	621	11051	25723	381	18.45	
21	SA	10	30	—	—	—	—	—	—	—	1
	PA			3	90	289	6000	11248	31	44.22	
	QA			3	63	81	1674	2544	61	33.27	
22	SA	30	30	—	—	—	—	—	—	—	1
	PA			6	179	4711	31721	83803	31	446.74	
	QA			5	1501	2276	67360	116143	871	54.56	
23	SA	20	40	—	—	—	—	—	—	—	1
	PA			4	100	391	15400	17680	41	97.78	
	QA			4	2281	1559	58235	126731	1326	53.22	
24	SA	6	50	[illegible]	[illegible]	172	6665	15615	1	5969.30	1
	PA			3	123	138	6200	9490	51	60.71	
	QA			3	101	104	2314	6264	101	30.99	
25	SA	10	50	—	—	—	—	—	—	—	1
	PA			3	136	317	13650	18794	51	113.66	
	QA			3	103	145	5000	7450	101	45.95	

SA: serial algorithm described in Section A.
PA: parallel algorithm described in Section B.
QA: parallel algorithm described in Section III.
m: number of constraints(not including nonnegativity constraints)
n: number of variables
Iter: number of iterations (not including initialization)
N: $\sum_{i=1}^{iter} N_a(i)$, $N_a(i)$ = number of active processors at i-th iter
Vmax: maximal number of vertices stored in memory
Vgen: maximal number of new vertices generated in one iter
Vtotal: total number of generated vertices by cutting
Time: approximated CPU-time (in seconds)
Sno: number of global solution

In Table 2, SA may be not run in some test problems because SA always requires a significantly large memory than the two other methods.

Essentially, the parallel algorithm introduced in Section II and Section III is an asynchronous parallel procedure; where processors can communicate with one another at all times. Although they are independent subproblems, the promising vertices of some subproblems may overlap some times. Notice that it can not guarantee that a particular number of parallel subproblems will need to be solved at each iteration (i.e. the number of active processors may vary anywhere from 0 to the maximal number at each iteration). For example, in the test problem no.12, the number of active processors in each iteration will be 11 (initial),10,10,6 for PA, and 21 (initial),10,9 for QA. Actually, it is possible that the inactive processors could share computing with the active ones which have more subproblems (i.e. more promising vertices) to be solved, and the percentage of processor utilization can be improved.

Finally, there is one point worth noting in the practical implementation of our algorithm, i.e. in some computational results of Table 2, QA with $n^2 + 1$ processors takes more time than SA with $n+1$ processors since the same vertex maybe appears repeatedly in some processors. In order to avoid too much overlapping in the computations of processors, the initial enclosing polyhedron must be carefully partitioned.

VI. Conclusion

In cutting plane algorithms for solving concave minimization problems the number of new vertices generated by cut in each iteration is rapidly increasing with n (cf., e.g., [2, 5, 6, 7, 12]). The serial algorithm in this paper, of course, requires much time in computing and a large memory in storage. For each processor in our parallel algorithm, however, we decrease both the number of new vertices generated by cut and the storage memory using the method of partition and the comparison of the upper bounds of all subproblems.

Computational results in Table 2 have shown the efficient performance of our parallel algorithm in minimizing a concave function over a polyhedron constraint set by the cutting plane method, due to the reasonably short computing time. In the future, we believe that parallel algorithm will play a significant role in solving the nonconvex optimization problems, especially in improving the performance of computing in large scale nonconvex optimization problems.

APPENDIX: In this Section, we will extend the method proposed in this paper to a problem of globally minimizing concave function over a general convex set. As we know, nonlinear convex set can be assumed to be constructed by infinite number of linear constraints. Therefore, we can apply the algorithm mentioned here to the problem of minimizing concave function subject to con-

Table 3: Iterative Results for Example 3

k		Processors (i) 1	2	3	S^*	f^*
0	$f(p_i)$	f(1.0094,0.8070) = −2.4615	f(1.2072,0.4024) = −2.8332	f(1.7236,1.8944) = −1.2320	{1,2,3}	−2.8332
	$f(v_i)$	f(3.2156,0.4024) = −3.6825	f(1.0094,2.6086) = −3.2104	f(1.0094,0.4024) = −3.1860		
1	λ	0.5439	0.7150	0.2455	{3}	−2.8332
	Constraint	ζ_5	ζ_1	ζ_4		
	$f(z_i^1)$	f(1.9908,0.8095) = −1.4769	f(1.2268,1.4832) = −1.5982	f(1.0840,0.5576) = −2.7271		
	Cutting Plane	$3.9267X_1 - 0.3811X_2 < 7.5090$	$-28X_1 + 9X_2 < -21$	$-53.2440X_1 - 36.0000X_2 < -77.7921$		
	New Vertices	(1.9513,0.4024) (2.0632,1.5549)	(1.4476,2.1704) (1.0094,0.8070)	(1.1890,0.4024) (1.0094,0.6680)		
	The Promising Vertices			(1.1890,0.4024)		
	Lower Bound	−2.2071	−2.4615	−2.8625		
2	λ			0.0255	{3}	−2.8332
	Constraint			ζ_4		
	$f(z_i^2)$			f(1.1921,0.4185) = −2.8223		
	Cutting Plane			$-39.4063X_1 - 36.0000X_2 < -62.0436$		
	New Vertices			(1.1381,0.4777) (1.2068,0.4024)		
	The promising Vertices			(1.2068,0.4024)		
	Lower Bound			−2.8338		
3	λ			5.1696×10^{-4}	{3}	−2.8332
	Constraint			ζ_4		
	$f(z_i^3)$			f(1.2069,0.4027) = −2.8330		
	Cutting Plane			$-37.5174X_1 - 36.0000X_2 < -59.7779$		
	New Vertices			(1.1995,0.4104) (1.2072,0.4024)		
	The promising Vertices			(1.2072,0.4024)		
	Lower Bound			−2.8332		
4	λ			2.3540×10^{-7}	ϕ	−2.8332
	Constraint			ζ_4		
	$f(z_i^4)$			f(1.2072,0.4024) = −2.8332		

vexset, and terminate until the enclosing polyhedron is sufficiently close to the feasible set, i.e. λ is sufficiently small. So, if we give an appropriate tolerence $\epsilon > 0$ for λ, then the algorithm will converge to the global solution in a finite iterations.

Consider the constrained global concave minimization problem

$$(\mathcal{PP}) \begin{cases} \text{minimize} & f(x) \\ \text{subject to} & D \bigcap G \end{cases}$$

where f is a concave function on R^n, $D = \{x \in R^n : Ax \leq b, \ x \geq 0\}$, $G = \{x \in R^n : g_i(x) \leq 0, \ i = 1, \dots, p\}$, g_i is a convex function on R^n whose gradient is continuous, A is an m×n matrix, $b \in R^m$, and $D \bigcap G \neq \phi$, bounded. Assume that there is a strict interior point in the feasible set. Now, we will apply the algorithm described in Section B.with the following modifications which incorporate the idea of Hoffman[9] to solve problem $(\mathcal{PP})$.

- Compute

$$\begin{cases} \min x_j (j = 1, \dots, n) \\ \text{s.t. } x \in D \bigcap G \end{cases} \begin{cases} \max \sum_{i=1}^n x_i \\ \text{s.t. } x \in D \bigcap G \end{cases} \quad (20)$$

by Kelley's method[13] with old cut deletion procedure proposed by Topkis[14] and Eaves and Zangwill[15].

- For the nonlinear constraints, we use the golden section algorithm to obtain λ.
- In the algorithm, the cutting plane is either $a_i x - b_i = 0$ for the active linear constraint or the linearization of $g_j(x)$ i.e. $g_j(z_i^k) + \nabla g_j(z_i^k)^T (x - z_i^k) = 0$ for nonlinear constraint $g_j(x)$.
- Given a tolerance number $\epsilon > 0$, if $\lambda < \epsilon$, then algorithm stops.

In order to illustrate how extend the method developed in this paper to compact convex sets, we give a small example as follow:

Example 3.

$$\begin{array}{lll} \text{minimize} & -(x_1 - 2)^2 - (x_2 - 1.5)^2 - 1 & \\ \text{subject to:} & -28x_1 + 9x_2 + 21 \leq 0, & \zeta_1 \\ & 9x_1^2 - 72x_1 + 16x_2^2 \leq 0, & \zeta_2 \\ & x_1^2 + x_2^2 - 16 \leq 0, & \zeta_3 \\ & 64x_1^2 - 192x_1 - 36x_2 + 153 \leq 0, & \zeta_4 \\ & x_1 - 3x_2 \leq 0, & \zeta_5 \\ & 4x_1^2 - 12x_1 + x_2^2 - 2x_2 + 9 \leq 0, & \zeta_6 \\ & x_1, x_2 \geq 0. & \end{array}$$

Applying Kelley's method in 3 processors, we have $p_1 = (1.0094, 0.8070)$, $p_2 = (1.2072, 0.4024)$, and $p_3 = (1.7236, 1.8944)$ yielding function values of -2.4615, -2.8332, and -1.2320 respectively. Therefore, $f^0 = -2.8332$, and we obtain the initial enclosing simplex D^0 with vertices $v_1 = (3.2156, 0.4024)$, $v_2 = (1.0094, 2.6086)$, and $v_3 = (1.0094, 0.4024)$. $N^0 = \phi$, since $f(v_1) = -3.6825$, $f(v_2) = -3.2104$, $f(v_3) = -3.1860 < f^0$, i.e., $S^0 = \{1,2,3\}$.

Let interior point $p = (1.3134, 1.0346)$ and $\epsilon = 10^{-6}$. Additional iterations are described in Table 3.

References

[1] T.H. Mattheiss: An Algorithm for Determining Irrelevant Constraints and all Vertices in Systems of Linear Inequalities. Operations Research 21, 247-260, 1973.

[2] R. Horst, N.V. Thoai and J. de Vries: On Finding New Vertices and Redundant Constraints in Cutting Plane Algorithms for Global Optimization. Operations Research Letters 7, 85-90, 1988.

[3] R. Horst and H. Tuy: Global Optimization. Berlin: Springer-Verlag 1990.

[4] T.H. Matheiss and D.S. Rubin: A Survey and Comparison of Methods for Finding all Vertices of Convex Polyhedral Sets. Mathematics of Operations Research 5, 167-185, 1980.

[5] J.E. Falk and K.L. Hoffman: A Successive Underestimation Method for Concave Minimization Problems. Mathematics of Operations Research 1, 251-259, 1975.

[6] J.E. Falk and K.L. Hoffman: Concave Minimization Via Collapsing Polytopes. Operations Research 34, 919-929, 1986.

[7] T.V. Thieu: Improvement and Implementation of Some Algorithms for Nonconvex Optimization Problems. Optimization – Fifth French-German Conference, Castel Novel 1988. Lecture Notes in Mathematics 1405, 159-170, Spring Verlag 1989.

[8] P.M. Pardalos and J.B. Rosen: Methods for Global Concave Minimization: A Bibliographic Survey. SIAM Review 28, 367-379, 1986.

[9] K.L. Hoffman: A Method for Globally Minimizing Concave Functions over Convex Sets. Mathematical Programming 20, 22-32, 1981.

[10] H. Tuy: Concave Programming under Linear Constraints. Dokl. Akad. Nauk. SSSR 159, 32-35, 1964. [Translated, Soviet Math. Dokl. 4, 1437-1440, 1964.]

[11] P.B. Zwart: Nonlinear programming: Counterexample to Two Global Optimization Algorithms. Operations Research 21, 1260-1266, 1973.

[12] R. Horst and N.V. Thoai: Modification, Implementation and comparison of Three Algorithms for Globally Solving Linearly Constrained Concave Minimization Problems. Computing 42, 271-289, 1989.

[13] J.E. Kelley: The Cutting Plane Method for Solving Convex Programs. SIAM. J. 8, 703-712, 1960.

[14] D.M. Topkis: Cutting Plane Methods without Nested Constraint Sets. Operations Research 18, 404-413 (1970).

[15] B.C. Eaves and W.I. Zangwill: Generalized Cutting Plane Algorithms. SIAM. J. Control 9, 529-542, 1971.