Journal of Global Optimization **24:** 149–161, 2002.

Fast Algorithm for the Cutting Angle Method of Global Optimization

L.M. BATTEN[1] and G. BELIAKOV[2]
[1]*School of Computing and Mathematics, Deakin University, 221 Burwood Hwy, Burwood 3125, Australia; Tel.: + 61-3-9251-7474; Fax: + 61-3-9251-7604; E-mail: lmbatten@deakin.edu.au*
[2]*School of Computing and Mathematics, Deakin University, 221 Burwood Hwy, Burwood 3125, Australia; Tel.: + 61-3-9251-7475; Fax: + 61-3-9251-7604; E-mail: gleb@deakin.edu.au*

Abstract. The cutting angle method for global optimization was proposed in 1999 by Andramonov et al. (Appl. Math. Lett. 12 (1999) 95). Computer implementation of the resulting algorithm indicates that running time could be improved with appropriate modifications to the underlying mathematical description. In this article, we describe the initial algorithm and introduce a new one which we prove is significantly faster at each stage. Results of numerical experiments performed on a Pentium III 750 Mhz processor are presented.

1. Introduction

Global optimization algorithms have been the subject of a great deal of interest in past years [5, 6, 11, 13]. With ever increasing computational speeds available, hope is growing that some long-standing difficult problems can be tackled using such algorithms. For example, the protein folding problem [5, 8] the phase problem in X-ray crystallography [7].

The cutting angle method for global optimization was proposed in 1999 in a series of articles by Rubinov et al. [1, 11, 12] and has subsequently been improved by these same authors in [2, 4, 11]. This paper focuses on the computer implementation of the algorithm as described in [2, 11]. We prove that a different approach to that implementation results in a new algorithm with significant improvement in running time. At the end of the paper we present comparative results on the execution time of both algorithms.

The cutting angle method is a deterministic optimisation method for Lipschitz functions in R^n [11]. A sequence of saw-tooth auxiliary functions, which underestimates the objective function, is built based on the support vectors for the given Lipschitz function. Thus, the Lipschitz optimization problem is transformed into a sequence of auxiliary problems to minimize the saw-tooth functions. It can be shown that any limit point of the sequence of corresponding solutions is a global minimum for the original Lipschitz function.

Minimization of the auxiliary functions is essentially a combinatorial problem, and thus grows exponentially if all possibilities are tested. An analogous problem is that of finding the intersection of n cones. Mladineo [9] suggested that to find this

intersection it was sufficient to consider sets of neighbouring cones, resulting in a major reduction in computations needed. A similar approach is taken in the present article. By a careful analysis of the situation, we are able to significantly reduce the number of possibilities tested.

2. The cutting angle method

The cutting angle method is based on results in abstract convexity [11]. The cutting angle method arises, as do the Piyavskii and Mladineo methods [6, 9, 14], as a special case of the generalized cutting plane method described in [11]. It involves a construction of the saw-tooth cover of the objective function $f(x)$, the max–min type auxiliary function that always underestimates $f(x)$. The maxima of the auxiliary function are taken at known values of $f(x)$.

The optization problem is then translated into a sequence of auxiliary problems of minimization of the saw-tooth cover. At every iteration, the global minimum of the auxiliary function is selected as the point where the objective function is evaluated next. The sequence of solutions of the auxiliary problems converges to a global minimum of $f(x)$.

This abstract formulation permits the simultaneous consideration of problems of arbitrary dimension, rather than a generalization of one-dimensional algorithms. Let $f(x)$ be a Lipschitz function defined on the unit simplex S. Consider the following problem of global optimisation

$$f(x) \to \min \text{ subject to } \quad x \in S\,.$$

It was shown in [1, 2, 11] that this problem can be reformulated as the global optimization problem of the so-called Increasing Positively Homogeneous Function of degree one, or IPH function, over the unit simplex. The class of IPH functions f defined on R^n_+ is

$$\{f\colon \forall x, y \in R^n_+ \;\; x \geqslant y \text{ implies } f(x) \geqslant f(y);\; \forall x \in R^n_+,\; \lambda \in R,\; \lambda > 0\colon \\ f(\lambda x) = \lambda f(x)\}\,,$$

where R^n_+ denotes the set of real vectors with non-negative components.

Here, and in the remainder of this paper, vector inequality $x \geqslant y$ means dominance, i.e., $\forall i : x_i \geqslant y_i$. Similarly $x > y$ means strict dominance, i.e., $\forall i : x_i > y_i$.

Examples of IPH functions are:

(1) $f(x) = a^t x,\; a_i \geqslant 0$;
(2) $f(x) = \|x\|_p,\; p > 0$;
(3) $f(x) = \Pi_{j\in J}\, x_j^{t_j},\; J \subset I = \{1, \ldots, n\},\; t_j > 0,\; \Sigma_{j\in J}\, t_j = 1$;
(4) $f(x) = \sqrt{[Ax, x]}$, where A is a matrix with nonnegative entries and $[\cdot, \cdot]$ is the usual inner product in R^n.

The following result can be found in [11] (p. 96). Let $g\colon S \to R$ be a positive

Lipschitz function defined on the unit simplex. Then it can be extended to a finite IPH function $f(x)$ (of degree $p \geqslant 1$) on the cone R^n_+, which would coincide with $g(x) + c$ on S. I.e., $f(x) = g(x) + c$ is an IPH function on the unit simplex with

$$c \geqslant 2L - \min_{x \in S} g(x)$$

where L is the least Lipschitz constant of g in the L_1-norm. Since adding a constant does not affect the location of the minima, we can effectively minimize any Lipschitz function on the unit simplex using this transformation with an appropriate constant.

Thus, without loss of generality, we consider the problem of minimization of an IPH function $f(x)$ over the unit simplex. For each $x \in R^n_+$ define the *support vector*

$$l = \left(\frac{f(x)}{x}\right) = \left(\frac{f(x)}{x_1}, \frac{f(x)}{x_2}, \ldots, \frac{f(x)}{x_n}\right).$$

Here we allow the components of the support vectors to take infinite values (if $x_i = 0$), which differs from the approach in [2], and we formally denote this by ∞. (This will not affect any mathematics of the method, we do this only for clarity of presentation.)

We will use n vectors $e^m = (0, \ldots, 0, 1, 0, \ldots, 0)$, with 1 in the mth position, and we call the corresponding support vectors $l^m = (f(e^m)/e^m)$, $m = 1, \ldots, n$, *basis vectors*.

We consider a set of $K \geqslant n$ support vectors (and hence K known values of the function $f(x)$ at K distinct points), $\mathcal{K} = \{l^k\}_{k=1}^K$. Let also the first n support vectors be the basis vectors (taken at the vertices of the simplex). This choice of support vectors guarantees that the algorithm will locate all local (and hence global) minimizers of the auxiliary function in the interior of the unit simplex.

The auxiliary function

$$h_K(x) = \max_{k \geqslant K} \min_{i=1,\ldots,n} l_i^k x_i$$

is the saw-tooth cover of $f(x)$. It always underestimates the value of $f(x) : h_K(x) \leqslant f(x)$. Hence, $\lambda_K = \min_{x \in S} h_K(x) \leqslant \min_{x \in S} f(x)$. On the other hand, the sequence of its minima, $\{\lambda_K\}_{K=n}^{\infty}$ is increasing [2, 11], and converges to the global minimum of $f(x)$ as proved in [11].

We can formulate the cutting angle algorithm as follows [11].

ALGORITHM 1.

Step 0. (Initialisation)

(a) Take points e^m, $m = 1, \ldots, n$, and construct basis vectors $l^m = (f(e^m)/e^m)$, $m = 1, \ldots, n$.

(b) Define the function $h_n(x) = \max_{k \leqslant n} \min_{i=1,\ldots,n} l_i^k x_i = \max_{k \leqslant n} l_k^k x_k$.

(c) Set $K = n$.

Step 1. Find $x^* = \arg[\min_{x \in S} h_K(x)]$.

Step 2. Set $K = K + 1$ and $x^K = x^*$.
Step 3. Compute $l^k = (f(x^K)/x^K)$. Define the function

$$h_K(x) = \max_{k \leq K} \min_{i=1,\ldots,n} l_i^k x_i = \max\{h_{K-1}(x), \min_{i=1,\ldots,n} l_i^K x_i\}.$$

Go to Step 2.

A more general version of this algorithm is known as the ϕ-bundle method, and its convergence under very mild assumptions was proven in [10].

The crucial and most time consuming step of the Algorithm 1 is Step 1, minimization of the auxiliary function. This problem is essentially of combinatorial nature. Some properties of the auxiliary function (1) are studied in [2]. Among them we note the following.

THEOREM 1 [2, 11]. *Let $x > 0$ be a local minimizer of $h_K(x)$ over the relative interior of S, $riS = \{x \in S, x > 0\}$. Then there exists a subset $L = \{l^{k_1}, l^{k_2}, \ldots, l^{k_n}\}$ of the set $\mathcal{K}$, such that*
(1) $x = (d/l_1^{k_1}, d/l_2^{k_2}, \ldots, d/l_n^{k_n})$ *with* $d = (\Sigma_i\, 1/l^{k_i})^{-1}$.
(2) $\max_{k \leq K} \min_{i=1,\ldots,n} l_i^k / l_i^{k_i} = 1$.
(3) *Either* $\forall i : k_i = i$, *or* $\exists m : k_m > n$, $l_i^{k_m} > l_i^{k_i}$, $\forall i \neq m$.

The value of the auxiliary function at x is $h_K(x) = d$.

REMARK. The authors of [4] also prove that $\forall i, k, 1 \leq i \leq n, n+1 \leq k \leq K : l_i^i \leq l_i^k$.

In order to find the global minimum of the auxiliary function at Step 2 of the algorithm, we need to examine all its local minima, and hence all combinations of the support vectors that satisfy the conditions of the Theorem 1.

This process can be significantly accelerated (as reported in [2]) by noticing, that

$$h_K(x) = \max\{h_{K-1}(x), \min_{i=1,\ldots,n} l_i^K x_i\}$$

Then, if we have already computed all the local minima of the auxiliary function $h_{K-1}(x)$ at the previous iteration, we only need to compute those minima that have been added by aggregation of the last support vector l^K. This means that we need to examine only those combinations of support vectors that include vector l^K (i.e., one of $l^{k_i} = l^K$). The cutting angle algorithm of [2, 11], which we improve here, works based on the above theorem, by examining all possible combinations of n support vectors (out of K).

3. Combinatorial formulation

In this section we translate the crucial step of the cutting angle method, minimization of the auxiliary function $k_K(x)$, into an abstract combinatorial problem of

selection of groups of n support vectors that satisfy certain conditions. Then we formulate our main theorem, which is the basis of the fast algorithm presented in the next section.

Consider a set of K support vectors $\mathscr{K} = \{l^k\}_{k=1}^K$, $l^k \in R_+^n$. Let I denote $\{1, 2, \ldots, \mathrm{n}\}$. From Theorem 1, the local minima of the auxiliary function $h_K(x)$ are combinations of n support vectors $L = \{l^{k_1}, l^{k_2}, \ldots, l^{k_n}\}$ that satisfy the following conditions:

(1) $\forall i,\ j \in I,\ i \neq j : l_i^{k_i} < l_i^{k_j}$

(2) $\forall \nu \in \mathscr{K} \backslash L,\ \exists i \in I : l_i^{k_i} \geqslant \nu_i$.

We call the subset L, which satisfies conditions (1) and (2) above, a *valid combination* of support vectors.

To illustrate these conditions, interpret L as a $n \times n$ matrix, whose rows are $l^{k_1}, l^{k_2}, \ldots, l^{k_n}$ [10]:

$$L = \begin{pmatrix} l_1^{k_1} & l_2^{k_1} & \ldots & l_n^{k_1} \\ l_1^{k_2} & l_2^{k_2} & \ldots & \\ \ldots & & & \\ l_1^{k_n} & \ldots & & l_n^{k_n} \end{pmatrix}.$$

We will denote the elements of this matrix by $K_{ji} = l_i^{k_j}$. Condition 1 implies that every element on the diagonal must be the smallest in its column, and condition 2 implies that for every vector ν of $\mathscr{K}$ that we take, not already in L, the diagonal of L is *not* dominated by ν: $\neg(diag(L) = (l_1^{k_1}, \ldots, l_n^{k_n}) < \nu)$.

As an example, consider the set $\mathscr{K} = \{(1, \infty, \infty),\ (\infty, 2, \infty),\ (\infty, \infty, 2),\ (2, 3, 4),\ (3, 4, 3)\}$. List the combinations satisfying (1):

$$L_1 = \{l^1, l^2, l^3\}, \qquad L_2 = \{l^4, l^2, l^3\}, \qquad L_3 = \{l^1, l^4, l^3\},$$

$$L_4 = \{l^1, l^2, l^4\}, \qquad L_5 = \{l^5, l^2, l^3\}, \qquad L_6 = \{l^1, l^5, l^3\},$$

$$L_7 = \{l^1, l^2, l^5\}, \qquad L_8 = \{l^4, l^2, l^5\}, \qquad L_9 = \{l^1, l^4, l^5\}.$$

Choose combinations satisfying (1) and (2):

$$L_4 = \{l^1, l^2, l^4\} = \begin{Bmatrix} 1 & \infty & \infty \\ \infty & 2 & \infty \\ 2 & 3 & 4 \end{Bmatrix}, \qquad L_5 = \{l^5, l^2, l^3\} = \begin{Bmatrix} 3 & 4 & 3 \\ \infty & 2 & \infty \\ \infty & \infty & 2 \end{Bmatrix},$$

$$L_6 = \{l^1, l^5, l^3\} = \begin{Bmatrix} 1 & \infty & \infty \\ 3 & 4 & 3 \\ \infty & \infty & 2 \end{Bmatrix}, \qquad L_8 = \{l^4, l^2, l^5\} = \begin{Bmatrix} 2 & 3 & 4 \\ \infty & 2 & \infty \\ 3 & 4 & 3 \end{Bmatrix},$$

$$L_9 = \{l^1, l^4, l^5\} = \begin{Bmatrix} 1 & \infty & \infty \\ 2 & 3 & 4 \\ 3 & 4 & 3 \end{Bmatrix}.$$

Let $\mathscr{L}^K$ denote the set of all valid combinations L of K support vectors satisfying conditions (1) and (2):

$$\mathscr{L}^K = \{L = \{l^{k_1}, l^{k_2}, \ldots, l^{k_n}\},\ l^{k_i} \in \mathscr{K}\colon\ (1)\ \forall i,\ j \in I,\ i \neq j : L_{ii} < L_{ji}$$
$$\text{and } (2)\ \forall \nu \in \mathscr{K} \backslash L\ \ \exists i \in I : L_{ii} \geqslant \nu_i\}$$

The problem of finding local minima of $h_K(x)$ is translated into the problem of listing the elements of $\mathscr{L}^K$. A simplistic approach is then:

Step 1. Construct all combinations L satisfying (1)
Step 2. Check the obtained combinations against (2).

The authors of [2, 11] then improve on this by taking into account the fact that all elements of $\mathscr{L}^K$ that do not involve l^K (i.e., minima of $h_{K-1}(x)$) do not need to be recomputed, hence the algorithm takes the form:

ALGORITHM 2.
Step 0. (*Initialisation*)
(a) Take points e^m, $m = 1, \ldots, n$, and construct basis vectors $l^m = (f(e^m)/e^m)$, $m = 1, \ldots, n$.
(b) Define the function $h_n(x) = \max_{k \leqslant n} \min_{i=1,\ldots,n} l_i^k x_i = \max_{k \leqslant n} l_k^k x_k$.
(c) Set $K = n$. Set $\mathscr{L}^K = \{\{l^1, l^2, \ldots, l^n\}\}$.
(d) Calculate $d = (\Sigma_{i=1,\ldots,n}\, 1/l_i^i)^{-1}$.
Step 1.
(a) Retrieve all valid combinations L(i.e., set $\mathscr{L}^K$).
(b) Select $L \in \mathscr{L}^K$ with the smallest d.
Step 2.
(a) $K = K + 1$.
(b) Take $x^* = d/\text{diag (L)}$ and evaluate $f(x^*)$.
(c) Compute $l^K = (f(x^*)/x^*)$. Define the function

$$h_K(x) = \max_{k \leqslant K} \min_{i=1,\ldots,n} l_i^k x_i = \max\{h_{K-1}(x), \min_{i=1,\ldots,n} l_i^K x_i\}.$$

Step 3.
(a) Check $\mathscr{L}^{K-1}$ against (2) and remove those that fail (2).
(b) Move the remaining combinations into $\mathscr{L}^K$.
Step 4.
(a) Construct all combinations L that involve l^K and satisfy (1).
(b) Calculate $d = (\Sigma_{i=1,\ldots,n}\, 1/l_i^{k_i})^{-1}$ for each such combination.
(c) Add these combinations to $\mathscr{L}^K$.
(d) Go to Step 1.

It is clear that at Step 4, the number of possible combinations L that formally need to be constructed and checked at each iteration K is $n\binom{K-1}{n-1}$, where $\binom{a}{b}$ denote binomial coefficients. Since $O(n)$ operations are needed to test condition (1), the complexity of the algorithm is $O(\binom{K-1}{n-1} n^2)$. Of course, in practice, fewer operations are needed: if, when forming L, condition 1 fails at half-way, there is no need to

complete the construction of this L in order to discard it. Still, the complexity of Step 4 is huge. The complexity of Step 3 of the alorithm is $O(|\mathscr{L}^{K-1}|Kn)$, where $|\mathscr{L}^{K-1}|$ is the cardinality of $\mathscr{L}^{K-1}$, i.e. the number of local minima of $h_{K-1}(x)$.

Next we formulate our main result.

THEOREM 2. *Let L satisfy conditions (1) and (2) and $K > n$. Then* $\forall j \in I,\ k_j \neq j$ $\exists \nu \in \mathscr{K}\backslash L : L^* = \{l^{k_1}, \ldots, l^{k_{j-1}},\ \nu,\ l^{k_{j+1}}, \ldots, l^{k_n}\}$ *satisfies conditions (1) and (2*)*: $\forall u \in \mathscr{K}\backslash(l^{k_j} \cup L^*)\ \exists i \in I : L^*_{ii} \geqslant u_i$.

Moreover, $diag(L^*) \leqslant l^{k_j}$.

Note that the difference between (2) and (2*) is that L^* is not tested if $u = l^{k_j}$ (i.e., condition (2*) is weaker than (2)).

Proof. If $K = n$, $\mathscr{L}^n$ consists of only one element, $\mathscr{L}^n = \{L\} = \{\{l^1, l^2, \ldots, l^n\}\}$ and $\mathscr{K}\backslash L = \emptyset$.

Let $K > n$. $\mathscr{K}\backslash(l^{k_j} \cup L^*) = \mathscr{K}\backslash(L \cup \nu) = (\mathscr{K}\backslash L)\backslash\nu$.

For a fixed j, consider the following subsets of $\mathscr{K}$:

$$\mathscr{V}_1 = \{u \in (\mathscr{K}\backslash L) : u_j \leqslant L_{jj} \text{ and } \forall i \in I,\ i \neq j : u_i > L_{ii}\},$$

$$\mathscr{V}_2 = \{u \in (\mathscr{K}\backslash L),\ \forall i \in I : u_i > L_{ii}\} \quad \text{and}$$

$$\mathscr{V}_3 = \{u \in (\mathscr{K}\backslash L),\ \exists i \in I,\ i \neq j : u_i \leqslant L_{ii}\}.$$

Clearly, $\mathscr{K}\backslash L = \mathscr{V}_1 \cup \mathscr{V}_2 \cup \mathscr{V}_3$. The set $\mathscr{V}_2$ is empty, because otherwise L would not satisfy condition (2). The set $\mathscr{V}_1$ is not empty, because it contains at least the jth basis vector l^j (note that since $l^{k_j} \neq l^j$, $l^j \notin L$. See also Remark after Theorem 1).

Take $\nu^* \in \mathscr{V}_1$ with the biggest jth component.

Then, by definition of $\mathscr{V}_1$, $L^* = \{l^{k_1}, \ldots, l^{j_{j-1}}, \nu^*, l^{k_{j+1}}, \ldots, l^{k_n}\}$, satisfies (1).

Let us demonstrate that L^* also satisfies (2*). Suppose, it does not. Then $\exists u \in (\mathscr{K}\backslash L)\backslash\nu^*,\ \forall i \in I : L^*_{ii} < u_i$. Let us establish which part of $\mathscr{K}\backslash L$ u can be chosen from. Firstly, $u \notin \mathscr{V}_3$, because it does not fit the definition of $\mathscr{V}_3$. Also $u \notin \mathscr{V}_1$ because this is how we chose $\nu^* : \forall u \in \mathscr{V}_1 : u_j \leqslant \nu^*_j = L^*_{jj}$. Lastly, since $\mathscr{V}_2$ is empty, no such u can be chosen from $\mathscr{K}\backslash L$, and therefore from $(\mathscr{K}\backslash L)\backslash\nu^*$, and hence L^* satisfies (2*).

Finally, if not $diag(L^*) \leqslant l^{k_j}$ for some $\nu^* \in \mathscr{V}_1$, then $L^*_{ii} > l^{k_i}_i$ for some i. Since except in row j, L^* coincides with L, and L satisfies (1), this is only possible when $i = j$. Therefore we must have $L^*_{jj} = \nu^*_j > l^{k_j}_j = L_{jj}$, but this implies that $\nu^* \in \mathscr{V}_2$, which is empty, hence contradiction. Proof completed.

4. Discussion

The first part of the theorem states that every valid combination of support vectors L

has (at least one) predecessor—an 'almost' valid combination L^*, which differs from L by only one vector l^{k_j}. This combination L^* would have been valid if not for l^{k_j} (hence the condition (2*) and not (2)). The second part of the theorem establishes the fact that predecessors of L cannot satisfy (2).

The major implication of Theorem 2 is that we can organize the set $\mathscr{L}$ of all $n \times n$ matrices L satisfying conditions (1) and (2) at some stage into a directed acyclic rooted graph, and hence are able to obtain all these elements by moving from one node of the graph to another, rather than trying all possible combinations of support vectors. Let us detail this process.

We start with $\mathscr{K} = \{l^k\}_{k=1}^n$ and will add one new support vector at a time. $\mathscr{L}^n = \{L_{root}\} = \{\{l^1, l^2, \ldots, l^n\}\}$. Add a new vector l^{n+1} and obtain $\mathscr{L}^{n+1}$. Then add another vector and obtain $\mathscr{L}^{n+2}$, etc. At every iteration K, $K = n+1, \ldots$, we need to find only those elements of $\mathscr{L}^K$, that contain the new vector l^K, since all the rest are already in $\mathscr{L}^{K-1}$.

By Theorem 2, every element of $L \in \mathscr{L}^K$, $K > n$, containing l^K at position j has a predecessor $L^* \in \mathscr{L}^{K-1}$, such that $diag(L^*) \leqslant l^K$. Then L is obtained from L^* by substituting the vector at position j with l^K. Let elements of $\mathscr{L}^n$, $\mathscr{L}^{n+1}, \ldots$ be nodes of the graph and let the arcs connect the nodes with their predecessors. Theorem 2 guarantees that there is no node without a predecessor, except L_{root}, and hence every node can be reached from the root by repeatedly substituting the appropriate support vectors. Hence the following corollary.

COROLLARY. *Define the directed acycic graph* $G = (V, E)$, *where* $V = \mathscr{L}$, $E = \{(L^*, L) : L^* \in \mathscr{L}^{K-1} \backslash \mathscr{L}^K,\ L \in \mathscr{L}^K \backslash \mathscr{L}^{K-1},\ K > n+1\}$. *Then the root* $r = L_{root}$ *is connected to every vertex of the graph.*

Thus, having proved that the whole set $\mathscr{L}^K$ of minima of the auxiliary function is on the graph, to find the elements of this set we only need to examine the nodes of the graph, and not all possible combinations of the support vectors.

We can interpret this in terms of the original optimization problem. At iteration $K-1$, the auxiliary function $h_{K-1}(x)$ has $|\mathscr{L}^{K-1}|$ local minima. The objective function is evaluated at a new point, and a new support vector l^K is added. The auxiliary function is modified (and becomes $h_K(x)$). Some of the local minima of $h_{K-1}(x)$ will disappear, and some new minima will appear. Theorem 2 demonstrates, that all the new minima of $h_K(x)$ can be obtained from those minima of $h_{K-1}(x)$ that have disappeared from $h_K(x)$, and also establishes the exact way the new minima can be found. Based on Theorem 2 we can design a fast algorithm that builds the minima of $h_K(x)$ from those of $h_{K-1}(x)$.

5. Fast algorithm

From the results of the previous section we derive the following new algorithm.

ALGORITHM 3.

Step 0.

(a) Evaluate the objective function $f(x)$ in the vertices of the unit simplex and construct n basis vectors $l^1, l^2, \ldots, l^n$.

(b) Build $\mathscr{L}^n = \{L_{root}\} = \{\{l^1, l^2, \ldots, l^n\}\}$,

(c) Calculate $d = (\Sigma_{i=1,\ldots,n}\, 1/l_i^i)^{-1}$.

(d) Take $K = n$.

Step 1.

(a) Select $L \in \mathscr{L}^K$ with the smallest d.

(b) Take $x^* = d/diag(L)$ and evaluate $f(x^*)$.

Step 2.

(a) Take $K = K + 1$.

(b) Form $l^K = (f(x^*)/x_1^*,\ f(x^*)/x_2^*, \ldots,\ f(x^*)/x_n^*)$.

Step 3.

(a) Select from $\mathscr{L}^{K-1}$ elements L with $diag(L) \leqslant l^K$. Call this set $\mathscr{L}^-$.

(b) $\mathscr{L}^K = \mathscr{L}^{K-1} \backslash \mathscr{L}^-$.

Step 4.

(a) For every $L \in \mathscr{L}^-$ and every $i = 1, 2, \ldots n$ replace l^{k_i} by l^K.

(b) Check these new combinations against condition (1) and if valid, calculate $d = (\Sigma_{i=1,\ldots,n}\, 1/l_i^{k_i})^{-1}$ for each of them.

(c) Add these valid combinations to $\mathscr{L}^K$.

Step 5. $\mathscr{L}^- = \emptyset$. Go to Step 1.

The stopping criterion is as in [2, 11].

Let us estimate the complexity of the Algorithm 3. At every iteration K, Step 3 takes at most $O(n|\mathscr{L}^{K-1}|)$ operations, and Step 4 takes $O(n^2|\mathscr{L}^-|)$ operations. $|\mathscr{L}^{K-1}|$ is the number of local minima of the auxiliary function $h_{K-1}(x)$, which depends on the objective function and K. This corresponds to the number of vertices of the saw-tooth cover of f. $|\mathscr{L}^-|$ is the number of local minima lost when adding l^K. Hence, the complexity of constructing a set of valid combinations $\mathscr{L}^K$ is reduced from $O(n^2\binom{K-1}{n-1})$ to $O(n^2|\mathscr{L}^-|)$. The complexity of checking combinations against condition (2) is reduced to $O(Kn|\mathscr{L}^{K-1}|)$ to $O(n|\mathscr{L}^{K-1}|)$.

Algorithm 3 is superior to Algorithm 2 in all cases. Notice that both algorithms perform exactly the same tests (conditions 1 and 2) on combinations of support vectors. However Algorithm 3 performs test of condition 1 on a small subset $\mathscr{L}^-$, whereas Algorithm 2 performs this test on all possible combinations. Algorithm 3 performs test 2 on $\mathscr{L}^{K-1}$ using only one support vector l^K, whereas Algorithm 2 uses all K vectors. The only trivial case when both methods behave the same is $K = n + 1$.

6. Examples

We performed several tests of our algorithm against the original cutting angle

algorithm from [2, 11]. We chose three IPH test functions (each in three, five and 10 variables), and three Lipschitz functions (two in three, five and 10 variables and the third in two variables). The test IPH functions are defined in [2]

$$f_1(x) = \max_{i=1,2,\ldots,n} \{a_i x_i\} + \min_{j=1,2,\ldots,n} \{b_j x_j\},$$

$$a_i = 2 + \frac{i}{2}; \qquad b_i = (i+2)(n-i+2), \quad i = 1, 2, \ldots, n.$$

$f_2(x) = \max_{i=1,2,\ldots,40} \{[a^i, x]\} + \min_{i=1,2,\ldots,20} \{[b^j, x]\}$, $[a, x]$ denotes scalar products,

components of a^i, b^j are given by

$$a_k^i = \frac{20i}{k(1+|i-k|)}; \qquad i = 1, 2, \ldots, 40; \quad b_k^j = 5|\sin(j)\sin(k)|,$$

$$j = 1, 2, \ldots, 20, \quad k = 1, 2, \ldots, n.$$

$$f_3(x) = \max_{i=1,\ldots,20} \min_{j=1,\ldots n} [a^{ij}, x],$$

$$a_k^{ij} = \frac{10j}{k(1+|k-j|)} |\cos(i-1)|, \quad i = 1, 2, \ldots, 20, \quad j = 1, 2, \ldots, n,$$

$$k = 1, 2, \ldots, n.$$

Lipschitz test functions:

$$f_4(x) = \sum_{i=1}^{n} \min\{0, 15\|x - a^i\|^2 - b_i\}, \quad x \in S_1$$

$$a_n^i = \begin{cases} (n+1)/2n, & \text{if } i = j \\ 1/2n, & \text{otherwise}, \end{cases} \quad b_i = 4, \quad b_i = b_{i-1} - 2/(n-1),$$

$$i = 1, 2, \ldots, n$$

Griewanks function:

$$f_5(x) = \frac{1}{d} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad d = 200; \quad -20 \leqslant x_i \leqslant 20$$

Six-hump camel back function:

$$f_6(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1 x_2 + 4(x_2^2 - 1)x_2^2, \quad -2 \leqslant x_i \leqslant 2$$

Because Lipschitz functions 5 and 6 were defined in a hypercube rather than simplex, the following coordinate transformation (from $(n+1)$-simplex to R^n) was used

$$y_i = \frac{\ln x_{i+1}}{\ln x_i}, \quad i = 1, \ldots, n.$$

This transformation is smooth one-to-one and in the interior of the unit simplex. Its inverse (from R^n to $(n+1)$ unit simplex) was computed recursively as

Table 1. Execution time (s) of the original and proposed algorithms; maximum number of support vectors $K = 100$

Test function	$n = 3$		$n = 5$		$n = 10$	
	Algorithm 2	Algorithm 3	Algorithm 2	Algorithm 3	Algorithm 2	Algorithm 3
f_1	0.37	0.01	24.66	0.01	208.9	0.06
f_2	0.32	0.01	21.98	0.01	156.81	0.07
f_3	0.33	0.01	97.09	0.02	1451.2	0.19
f_4	0.33	0.01	36.12	0.02	318.2	0.08
f_5	0.36	0.01	78.11	0.02	594.4	0.11

$$x_1 = \frac{1}{Z};$$
$$x_2 = x_1 e^{y_1};$$
$$x_3 = x_1 e^{y_1 + y_2};$$
$$\ldots$$
$$x_{n+1} = x_1 e^{\sum_{i=1}^{n} y_i},$$

where the normalization constant $Z = 1 + \sum_{i=1}^{n} (e^{\sum_{j=1}^{i} y_j})$.

Numerical experiments were performed on a Pentium III 750 Mhz processor. The results are presented in Tables 1–3.

Table 2 warrants some comments. The dependency of the execution time on the number of support vectors is presented graphically in Figure 1. It appears that for all choices of the number of variables n, this dependency is polynomial ($\sim K^{2.5}$). On the logarithmic scale, all plots are parallel straight lines. This indicates that at least for up to 40 variables, the new algorithm will be able to produce results in a reasonably

Table 2. Execution time (s) of the Algorithm 3 as a function of the number of support vectors; test function f_1

K	$n = 5$	$n = 10$	$n = 20$	$n = 30$	$n = 40$
100	0.01	0.06	0.07	0.11	0.15
300	0.22	0.89	1.43	2.32	3.18
500	1.13	3.66	4.97	7.10	9.98
700	2.63	8.82	10.60	14.72	20.48
1000	6.01	20.40	24.46	32.54	43.97
1300	11.02	37.04	49.63	58.74	78.18
1500	15.39	51.14	72.94	81.48	107.76
1700	20.58	67.72	101.65	108.82	143.54
2000	30.05	96.85	152.70	160.79	209.48

Table 3. Execution time of the Algorithm 3 (s), number of support vectors taken, number of minima of the auxiliary function and the precision of the solution for Lipschitz test cases. Note that all minima found by the cutting angle method are within the radius of convergence of local descent algorithms (e.g., Newton's method, or discrete gradient method [3]), which can be used to improve the precision

Function	Global minimum	Minimum found	Execution time (s)	Number of support vectors	Number of minima of $h_k(x)$
f_5, $n = 2$	(0, 0)	(−0.025, 0.006)	0.821	2000	1945
f_5, $n = 3$	(0, 0, 0)	(0.012, 0.02, −0.018)	12.3	4500	12113
f_5, $n = 5$	(0, 0, 0, 0, 0)	(0.012, −0.033, −0.062, −0.031, −0.018)	1210	10000	311443
f_6, $n = 2$	(±0.08985, ∓0.71265	(−0.0882, 0.713)	6.13	4000	8252

short and predictable time. Given the upper bound on the number of iterations, computing time can be estimated from such a dependency.

7. Conclusion

We have propsed a substantially improved algorithm for the cutting angle method of global optimisation. Our algorithm reduces the complexity of the crucial step of the cutting angle method: minimization of the auxiliary max–min function. We formulated this step as an abstract combinatorial problem, and have demonstrated that all local minima of the auxiliary function are nodes of a directed graph with a

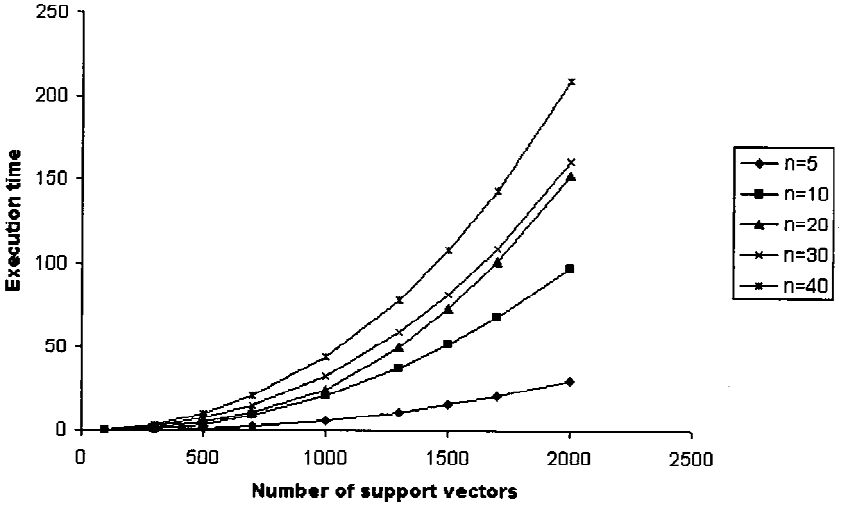

Figure 1. Execution time of Algorithm 3 as a function of the number of support vectors.

single root. Listing the nodes of the graph is a much more efficient operation than checking all possible combinations of support vectors, hence a dramatic improvement in computation time.

References

[1] Andramonov, M., Rubinov, A. and Glover, B. (1999), Cutting Angle Method in Global Optimization, *Appl. Math. Lett.* 12, 95–100.
[2] Bagirov, A. and Rubinov, A. (2000), Global Minimization of Increasing Positively Homogeneous Functions over the Unit Simplex, *Annals of Operations Res.* 98, 171–187.
[3] Bagirov, A. (1999), Derivative-free Methods for Unconstrained Nonsmooth Optimization and its Numerical Analysis, *Journal Investigacao Operacional* 19, 75–93.
[4] Bagirov, A. and Rubinov, A. (2001), Modified Versions of the Cutting Angle Method, in Hadjisavvas, N. and Pardalos, P.M. (eds.), *Convex Analysis and Global Optimization*, Vol. 54, Kluwer, Dordrecht.
[5] Floudas, C. (2000), *Deterministic Global Optimization. Theory, Methods and Applications*, Kluwer, Dordrecht.
[6] Hansen, P. and Jaumard, B. (1995), Lipschitz Optimization, in Horst, R. and Pardalos, P. (eds.), *Handbook of Global Optimization*, Kluwer, Dordrecht, 407–493.
[7] Hauptman, H. (1996), A Minimal Principle in the Phase Problem of X-Ray Crystallography, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 23, AMS.
[8] Maranas, C.D., Androulakis, I.P. and Floudas, C. (1996), A Deterministic Global Optimization Approach for the Protein Folding Problem, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 23, AMS, 133–150.
[9] Mladineo, R. (1986), An algorithm for Finding the Global Maximum of a Multimodal, Multivariate Function, *Math. Progr.* 34, 188–200.
[10] Pallachke, D. and Rolewicz, S. (1997), *Foundations of Mathematical Optimzation* (*Convex Analysis with Linearity*), Kluwer, Dordrecht.
[11] Rubinov, A. (2000), *Abstract Convexity and Global Optimization*, Kluwer, Dordrecht.
[12] Rubinov, A. and Andramonov, M. (1999), Lipschitz Programming via Increasing Convex-along-rays Functions, *Optimization Meth. and Software* 10, 763–781.
[13] Torn, A. and Zilinskas, A. (1989), *Global Optimization*, Springer, Heidelberg.
[14] Pijavski, S.A. (1972), An Algorithm for Finding the Absolute Extremum of a Function, *USSR Comput. Math. and Math. Phys.* 2, 57–67.