A Deterministic Algorithm for Global Optimization

By

Leo Breiman, Univ. of California, Berkeley*
Adele Cutler, Utah State University

Technical Report No. 224
October 1989

*Partially supported by NSF DMS-8718362

Department of Statistics
University of California
Berkeley, California

A Deterministic Algorithm for Global Optimization

LEO BREIMAN, *University of California, Berkeley* *
ADELE CUTLER, *Utah State University*

We present an algorithm for finding the global maximum of a multimodal, multivariate function for which derivatives are available. The algorithm assumes a bound on the second derivatives of the function and uses this to construct an upper bound surface. Successive function evaluations lower this surface until the value of the global maximum is known to the required degree of accuracy. The algorithm has been implemented in RATFOR and execution times for standard test functions are presented at the end of the paper.

1. Introduction

The problem considered is that of finding $f_\star$, the unconstrained maxima of a differentiable function $f : \mathrm{X} \to \mathrm{R}^1$ where $\mathrm{X} \subset \mathrm{R}^m$ is a compact polytope.

We develop a space-covering deterministic algorithm similar to the algorithm studied by Shubert [21] and Mladineo [14], who assume that the objective function satisfies a Lipschitz condition. Instead we assume that the function satisfies the condition

$$f(\mathbf{x}) \quad \leq \quad f(\mathbf{y}) + \nabla f(\mathbf{y})'(\mathbf{x}-\mathbf{y}) + K\| \mathbf{x}-\mathbf{y} \|^2 \quad \text{for any } \mathbf{x},\mathbf{y} \in \mathrm{X} \tag{1}$$

where K is a known constant. This condition, in more than one dimension, leads to simpler geometric properties than the Lipschitz bound.

The idea of the algorithm is that each time the function is evaluated we have more information about f. In particular, if we evaluate the function at a point $\mathbf{x}_i$ where f is small, we know that the global maximum cannot be close to $\mathbf{x}_i$. With a large enough number of function evaluations the value of the global maximum can be determined to any required degree of accuracy.

A conceptual description of the algorithm is given in section 2, illustrated by a 1-dimensional example. In the 1-dimensional case, the algorithm consists of a simple updating procedure after each function evaluation. We find that the algorithm concentrates on the global maximum of the function and pays very little attention to local maxima. The third section extends the 1-dimensional algorithm to higher dimensions, developing the geometrical results used in implementing the algorithm. A detailed description of the algorithm is given in section 4.

In section 5 we use a simple example to illustrate the dependence of the algorithm on dimension. Then we present some timing results for the algorithm, along with a comparison of its performance with published results for other optimization methods tested on functions defined in the appendix.

The results are mixed. However, the present algorithm is guaranteed to find the global optimum. Many of the published results for other algorithms are vague about the proportion of times their

*Partially supported by NSF DMS-8718362

algorithm failed to find the global maximum. In addition, we have found that combining the present algorithm with an occasional local search leads to a drastic reduction in computing time (see Cutler [6]). This modification will be discussed in a future paper.

2. Background

Piyavskii [15] described a general approach to global optimization requiring continuous functions h_i defined on X such that for any $\mathbf{x}_i \in X$,

$$\begin{aligned} f(\mathbf{x}) &\leq h_i(\mathbf{x}) \quad \text{for all } \mathbf{x} \in X, \\ f(\mathbf{x}_i) &= h_i(\mathbf{x}_i). \end{aligned}$$

Suppose the function has already been evaluated at $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The *upper envelope at the n^{th} stage* is defined to be

$$B^{(n)}(\mathbf{x}) = \min_{i=1,\ldots,n} h_i(\mathbf{x}). \tag{2}$$

Now by assumption, $f(\mathbf{x}) \leq h_i(\mathbf{x})$, for any i, so

$$\begin{aligned} f_\star &= \max_{\mathbf{x}\in X} f(\mathbf{x}) \\ &\leq \max_{\mathbf{x}\in X}\left(\min_{i=1,\ldots,n} h_i(\mathbf{x})\right) \\ &= \max_{\mathbf{x}\in X} B^{(n)}(\mathbf{x}). \end{aligned}$$

Thus, a lower bound $\underline{f}_n$ and an upper bound $\bar{f}_n$ for $f_\star$ are given by

$$\underline{f}_n = \max_{i=1,\ldots,n} f(\mathbf{x}_i) \leq f_\star \leq \max_{\mathbf{x}\in X} B^{(n)}(\mathbf{x}) = \bar{f}_n.$$

Then Piyavskii proposes to determine $\mathbf{x}_{n+1}$, the location of the next function evaluation, by maximizing $B^{(n)}$, that is

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in X} B^{(n)}(\mathbf{x}).$$

It should be noted that the problem of maximizing $B^{(n)}$ and updating $B^{(n)}$ to $B^{(n+1)}$ is potentially as difficult as the original problem of maximizing f. Shubert [20] and Mladineo [13] assume that for all $\mathbf{x}, \mathbf{y}$

$$f(\mathbf{x}) \leq f(\mathbf{y}) + K\|\mathbf{x} - \mathbf{y}\|$$

for a known constant K leading to the bounding functions

$$h_i(\mathbf{x}) = f(\mathbf{x}_i) + K\|\mathbf{x} - \mathbf{x}_i\|.$$

Mladineo [14] treated the multidimensional case. The analytic difficulties involved forced her to use approximations of unknown accuracy in evaluating $\mathbf{x}_{n+1}$ and updating $B^{(n)}$.

Under assumption (1) we use the bounding functions

$$h_i(\mathbf{x}) = f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)'(\mathbf{x} - \mathbf{x}_i) + K\|\mathbf{x} - \mathbf{x}_i\|^2.$$

Surprisingly, this leads to simple and exact evaluations of $\mathbf{x}_{n+1}$ and updates for $B^{(n)}$.

2.2 One-dimensional Illustration

Let $f : \mathrm{R}^1 \to \mathrm{R}^1$ be given by

$$f(x) = \cos(cx) - x^2$$

with $\mathrm{X} = [-.2, 1]$ and $c = 5\pi$. The function has a global maximum at $x = 0$ and two local maxima, one at $x = .4$ and the other at $x = .8$. For this example $K = \frac{c^2}{2} - 1$.

If the function has been evaluated at $x_1, \ldots, x_n$ we have

$$\begin{aligned} h_i(x) &= f(x_i) + f'(x_i)(x - x_i) + K(x - x_i)^2 \\ &= Kx^2 + (f'(x_i) - 2Kx_i)x + (f(x_i) - f'(x_i)x_i + Kx_i^2) \\ &= Kx^2 + \beta_{i1}x + \beta_{i2}. \end{aligned}$$

The upper envelope $\mathrm{B}^{(n)}$ consists of n quadratic pieces intersecting at local maxima or cusps. One intersection occurs between each pair of adjacent function evaluations. The function is then evaluated at x_{n+1}, the largest intersection peak, introducing a new quadratic which intersects $\mathrm{B}^{(n)}$ in two places, one on each side of x_{n+1}.

Suppose we save all the function and gradient values and keep a list of the locations and heights of the intersections of $\mathrm{B}^{(n)}$, recording which function evaluations are directly to the left and right of each intersection. Then we can locate x_{n+1} by finding the intersection point with the largest value of $\mathrm{B}^{(n)}$ and removing it from the list, noting which function evaluations points are to either side of x_{n+1}. Denote the latter by x_j and x_k. Once the function has been evaluated at x_{n+1} and its value and that of the gradient have been saved, we must locate the two new intersection points, v_1 and v_2 and save them. But these can be found by solving $h_j(v_1) = h_{n+1}(v_1)$ and $h_k(v_2) = h_{n+1}(v_2)$, which is easy since we have saved all function values and derivatives. Finally we note that v_1 now has x_j on one side and x_{n+1} on the other and similarly, v_2 has x_k on one side, x_{n+1} on the other, so we can record all the information we need about the new intersection points.

For simplicity, we have ignored the endpoints. If the first function evaluation is made at an endpoint, the second will also, and after that the situation is as described above.

The algorithm's progress in lowering the upper envelope has been plotted in Fig. 1. The dashed lines denote the upper envelope $\mathrm{B}^{(n)}$. We note that in the later stages, the upper bound function is very close to f near the global max but not very close elsewhere. In particular, unlike almost all of its competitors, this algorithm pays little attention to the local maxima of f.

3. The General Algorithm

Implementation of the algorithm involves finding $\mathbf{x}_{n+1}$, the global maximum of $\mathrm{B}^{(n)}$, evaluating $f(\mathbf{x}_{n+1})$, and updating $\mathrm{B}^{(n)}$. In the 1-dimensional case this was easy because we could keep track of the intersections of $\mathrm{B}^{(n)}$ and update them. In this section the simple geometry of the one-dimensional case is extended to higher dimensions. The results are simple methods for finding $\mathbf{x}_{n+1}$ and updating $\mathrm{B}^{(n)}$.

Consider the situation after n function evaluations. Each point $\mathbf{x}_i, i = 1, \ldots, n$ has an upper bound function $h_i(\mathbf{x})$ associated with it. We know that $\mathrm{B}^{(n)}$ is equal to h_i at the point $\mathbf{x}_i$, and by continuity this equality will hold in a closed set including $\mathbf{x}_i$. Call this set $\mathrm{C}_i{}^{(n)}$, the *territory* of $\mathbf{x}_i$.

The territories are polytopes which cover X. This is illustrated in Fig. 2, which shows the function evaluations and the polytopes $\mathrm{C}_i{}^{(n)}$ for $n = 20$ evaluations of the COS2 function defined in the appendix. The function evaluations, $\mathbf{x}_1, \ldots, \mathbf{x}_{20}$ are marked with stars, the vertices with

Figure 1

Lowering of Upper Envelope, One-dimensional Example

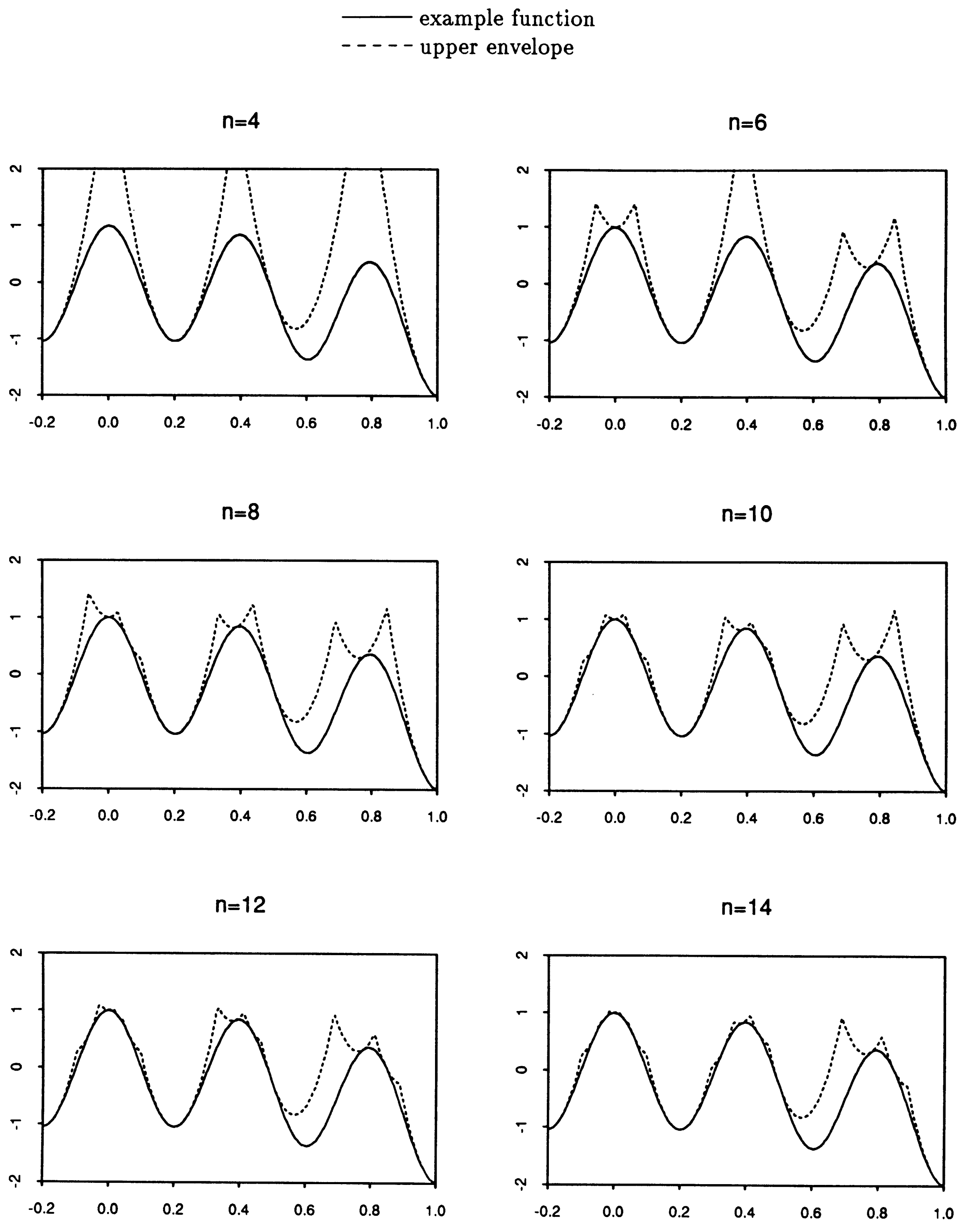
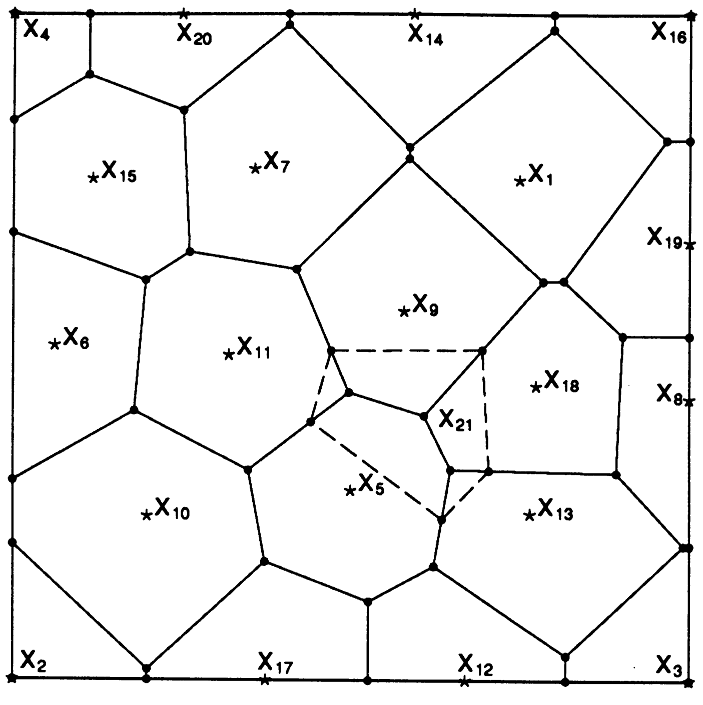


Figure 2

Vertex Structure for COS2 Example, $n = 20$



dots. The lines drawn are the edges of the sets $C_i^{(n)}$. We say that two vertices are *adjacent* if they have an edge between them.

This structure is particularly useful because it can be shown that a local maximum of $B^{(n)}$ can only occur at a vertex, so x_{n+1} can be found by searching through a list of vertices to find that vertex $v_\star$ for which $B^{(n)}$ is the largest. The function is then evaluated at $x_{n+1} = v_\star$ and the list of vertices is updated. This involves removing from the list any vertices v for which $B^{(n)}(v) > h_{n+1}(v)$, since these can not be vertices of $B^{(n+1)}$. Such vertices will be called *dead* vertices, the rest *alive* vertices. Once the dead vertices have been removed the new vertices, those of the polytope $C_{n+1}{}^{(n+1)}$, must be added to the list.

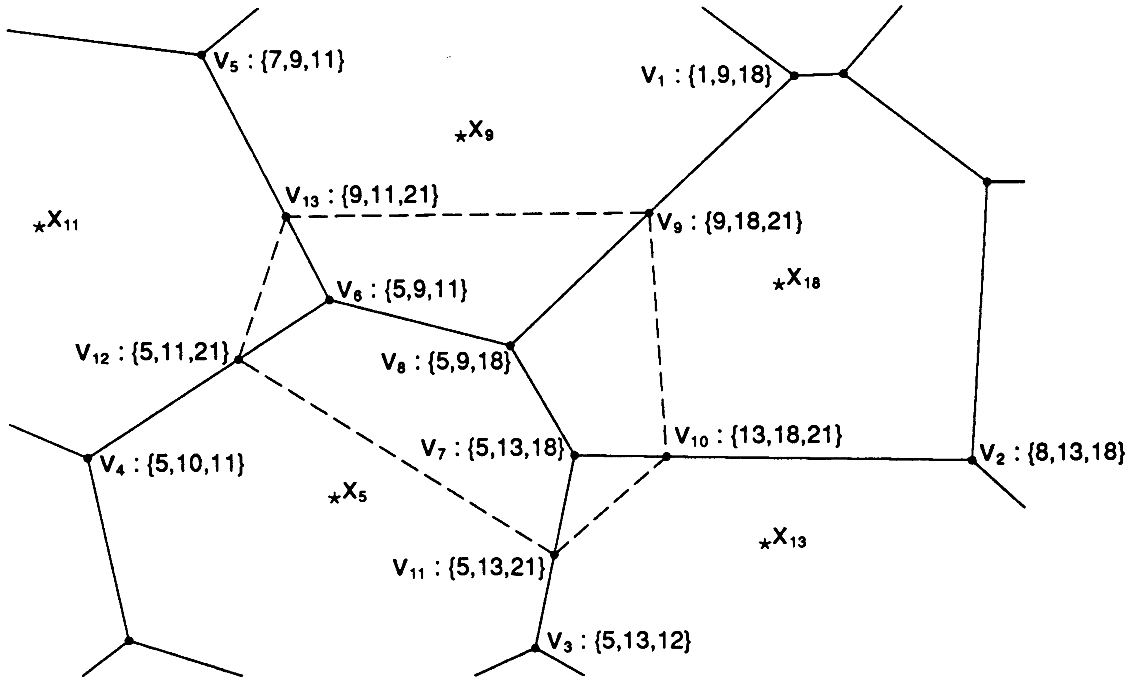
This procedure is illustrated in Figures 3 and 4. The vertex $v_\star$ has been marked on Fig. 3, and the edges of the new polytope C_{21} are drawn with dashed lines. Along the dashed lines we know that h_{21} is equal to h_j, the upper bound function for some neighboring point x_j. The vertices inside the dashed-line polytope are dead and will be removed, the vertices of the dashed-line polytope itself will be added, with lines between them as appropriate. This gives the updated vertex network given in Fig. 4.

To implement this updating procedure we have to be able to store the function and vertex information in a way that is easy to update. All function values and gradients will be stored since we don't know in advance when they will be needed. We also need to store the location of the vertices and the current value of $B^{(n)}$ at each. For every vertex a list of the $m + 1$ adjacent vertices will be needed and we will also store a list of which x_i's surround each vertex, which we call the *index set* of the vertex. In Fig. 4 the index set of each vertex is written beside it.

It will be shown that the dead vertices are connected via adjacencies (in the graph-theoretic sense), so we do not have to evaluate h_{n+1} at every vertex to find all the dead vertices. We can

Figure 3

New Polytope C_{21}



start at $\mathbf{v}_\star$ and move along the edges until all the dead vertices have been located, stopping each path once an alive vertex is found. Referring to Fig. 4, this involves starting at $\mathbf{v}_8 = \mathbf{v}_\star$ and finding the vertices adjacent to $\mathbf{v}_8$. These are $\mathbf{v}_1, \mathbf{v}_6$ and $\mathbf{v}_7$. The value of h_{n+1} at each of these vertices is calculated and compared to $B^{(n)}$ to determine that $\mathbf{v}_6$ and $\mathbf{v}_7$ are dead and $\mathbf{v}_1$ is alive. Then the vertices adjacent to $\mathbf{v}_6$ and $\mathbf{v}_7$ are examined. Since none of these vertices are dead, the algorithm stops; there can be no more dead vertices. If some of these vertices had been alive, their adjacent vertices would have been examined, and so on until no new dead vertices could be found adjacent to those already examined. One thing to note is that we only evaluate h_{n+1} at the vertices we encounter. The values are temporarily saved for use in determining the new vertices (see below).

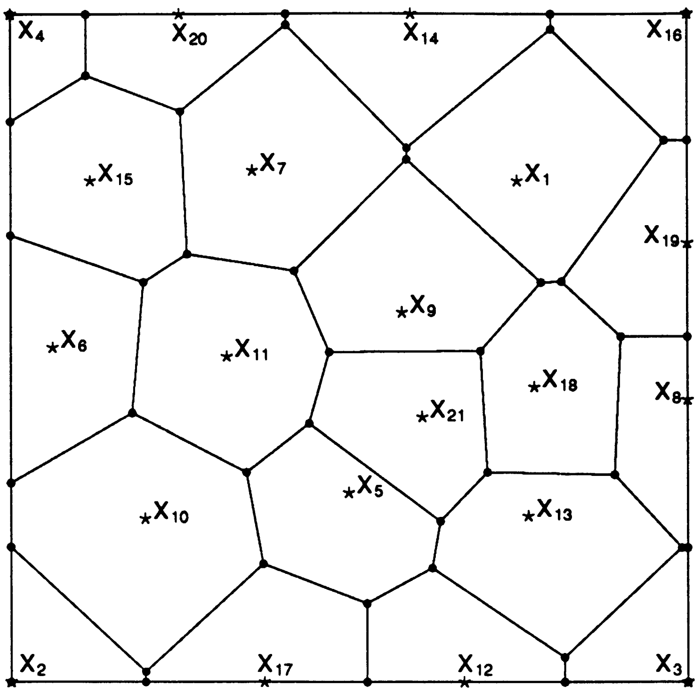
Referring again to Fig. 4, we see that each new vertex occurs on an edge between a dead vertex and an alive vertex, and every such pair gives rise to one new vertex. This turns out to be true in general and we refer to the adjacent pair as the *parents* of the new vertex. All the parents will have been examined while finding the dead vertices, but the position of the each new vertex still has to be determined. This can be done using information from the parent vertices and the values of h_{n+1} saved above.

The only tasks remaining are those of updating the index sets and adjacencies. The index sets of a vertex never change, so all that is required here is the creation of an index set for each new vertex. From Fig. 4, it can be seen that the index set of any new vertex can be obtained by taking the two indices common to both the dead and alive parent vertices and adding the new index "21" to the set. For example, the new vertex $\mathbf{v}_9$ is between the dead vertex $\mathbf{v}_8$, with index set $\{5, 9, 18\}$ and the alive vertex $\mathbf{v}_1$, with index set $\{1, 9, 18\}$. The index set of the new vertex is $\{9, 18, 21\}$.

For updating the adjacencies, note that the only existing adjacencies that change are those of the alive parents, each of which is now adjacent to a new vertex instead of a dead one. For example,

Figure 4

Vertex Structure for $n = 21$



in Fig. 4 the alive parent $\mathbf{v}_5$ is now adjacent to the new vertex $\mathbf{v}_{13}$ instead of the dead vertex $\mathbf{v}_6$. Conversely, each new vertex is adjacent to its alive parent. All that remains is to determine the adjacencies of the new vertices among themselves. First note that from Fig. 4 it seems that new vertices with the *same* dead parent must be adjacent. Take, for example, $\mathbf{v}_{12}$ and $\mathbf{v}_{13}$, both of which have the dead parent $\mathbf{v}_6$. This turns out to be true in general, and we show that there are, in fact, ways to determine all the new adjacencies without having to inspect every possible adjacent pair and count the number of elements in the intersection of the index sets.

The terminology of vertices, territories, etc. is used by Bowyer [2] and Watson [25] in the context of Dirichlet tesselation, which has a structure similar to the polytope tesselation considered here. In particular, the algorithm by Bowyer for adding a new point to a Dirichlet tessellation is close to the one we use.

The rest of this section contains more precise formulations of the geometrical results just described, culminating in a detailed description of the algorithm.

3.1 Definitions and Notation

Assume X is a compact subset of R^m and can be written $\mathrm{X} = \{\mathbf{x} \in \mathrm{R}^m : \mathbf{Ax} \leq \mathbf{b}\}$, for some $\mathbf{A}, \mathbf{b}$. With h_i defined as above, let $\mathrm{C}_i{}^{(n)}$, the *territory* of $\mathbf{x}_i$, be the set of points at which the i^{th} upper bound function is the smallest;

$$\mathrm{C}_i{}^{(n)} = \{\mathbf{x} \in \mathrm{X} : h_i(\mathbf{x}) \leq h_j(\mathbf{x}), j = 1, \ldots, n\},$$

and note that any $\mathbf{x} \in \mathrm{X}$ is a member of at least one $\mathrm{C}_i{}^{(n)}$. If we let

$$s_i = f(\mathbf{x}_i) - \mathbf{x}_i' \nabla f(\mathbf{x}_i) + K \| \mathbf{x}_i \|^2,$$

$$\mathbf{t}_i = \nabla f(\mathbf{x}_i) - 2K\mathbf{x}_i$$

we have $h_i(\mathbf{x}) = s_i + \mathbf{t}_i'\mathbf{x} + K\|\mathbf{x}\|^2$, so $h_i(\mathbf{x}) = h_j(\mathbf{x})$ iff $(\mathbf{t}_i' - \mathbf{t}_j')\mathbf{x} = s_j - s_i$. Letting $\mathbf{A}_i^{(n)} = (\mathbf{A}', \mathbf{t}_i - \mathbf{t}_1, \mathbf{t}_i - \mathbf{t}_2, \ldots, \mathbf{t}_i - \mathbf{t}_n)'$ and $\mathbf{b}_i^{(n)} = (\mathbf{b}', s_1 - s_i, s_2 - s_i, \ldots, s_n - s_i)'$ we have $\mathrm{C}_i^{(n)} = \{\mathbf{x} \in \mathrm{R}^m : \mathbf{A}_i^{(n)}\mathbf{x} \leq \mathbf{b}_i^{(n)}\}$ which gives the result below.

Proposition 3.1. *For $i = 1, \ldots, n$, $\mathrm{C}_i^{(n)}$ is a polytope.*

Definition 3.1 (vertex, vertex set, boundary vertex). *A vertex of $\mathrm{C}_i^{(n)}$ is an element of $\mathrm{C}_i^{(n)}$ which satisfies m linearly independent constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$. Let $\mathrm{V}^{(n)}$ be the set of all such vertices*

$$\mathrm{V}^{(n)} = \{\mathbf{v} : \mathbf{v} \textit{ is a vertex of } \mathrm{C}_i^{(n)} \textit{for some } i\}.$$

The constraints $\mathbf{Ax} = \mathbf{b}$ will be called boundary constraints, the others functional constraints since they have the form $h_i(\mathbf{x}) = h_j(\mathbf{x})$ for some j. A vertex for which the m linearly independent constraints are all boundary constraints will be called a boundary vertex.

Linear Independence Assumption *Assume that for every n, and for $i = 1, \ldots, n$, any $\mathbf{x} \in \mathrm{C}_i^{(n)}$ satisfies at most m constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$. In particular, any vertex satisfies exactly m constraints.*

This assumption will hold provided each boundary vertex satisfies exactly m linearly independent constraints from $\mathbf{Ax} = \mathbf{b}$, and for each n, and $h_{n+1}(\mathbf{v}) \neq \mathrm{B}^{(n)}(\mathbf{v})$ for every $\mathbf{v} \in \mathrm{V}^{(n)}$. This holds almost invariably in practice.

Definition 3.2 (index set). *For $\mathbf{x} \in X$ define an index set*

$$\mathrm{I}^{(n)}(\mathbf{x}) = \{-j : \mathbf{a}_j'\mathbf{x} = \mathbf{b}_j\} \cup \{j : h_j(\mathbf{x}) = \mathrm{B}^{(n)}(\mathbf{x})\}.$$

The negative elements of the index set refer to the boundary constraints satisfied by $\mathbf{x}$. These constraints will always be satisfied, so the negative elements of $\mathrm{I}^{(n)}$ will never change. Each positive element j refers to a polytope $\mathrm{C}_j^{(n)}$ of which $\mathbf{x}$ is a member. These will change whenever a new polytope is created which contains $\mathbf{x}$. The following Lemma gives a useful characterization of a vertex.

Lemma 3.1. *Let $\mathbf{x} \in X$. Then $\mathbf{x} \in \mathrm{V}^{(n)}$ iff $\mathrm{I}^{(n)}(\mathbf{x})$ contains exactly $m + 1$ elements.*

Proof. If $\mathbf{v} \in \mathrm{V}^{(n)}$ then $\mathbf{v} \in \mathrm{C}_i^{(n)}$ for some i, so $i \in \mathrm{I}^{(n)}(\mathbf{v})$ and $\mathbf{v}$ satisfies m linearly independent constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$. Each constraint contributes one element to the index set, giving a total of $m + 1$ elements. Conversely, $\mathbf{x}$ must be in $\mathrm{C}_i^{(n)}$ for some i since the polytopes cover X. There are m other elements of $\mathrm{I}^{(n)}(\mathbf{x})$. Each negative element represents a boundary constraint satisfied by $\mathbf{x}$, each positive one a functional constraint, so by the linear independence assumption $\mathbf{x}$ is a vertex of $\mathrm{C}_i^{(n)}$. ∎

Lemma 3.2. *If $\mathbf{v}$ is a vertex and $\mathbf{v} \in \mathrm{C}_i^{(n)}$, then $\mathbf{v}$ is a vertex of $\mathrm{C}_i^{(n)}$.*

Proof. If $\mathbf{v} \in \mathrm{C}_i^{(n)}$ then $i \in \mathrm{I}^{(n)}(\mathbf{v})$. The m other elements of $\mathrm{I}^{(n)}(\mathbf{v})$ correspond to m constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$ that are satisfied by $\mathbf{v}$ so by the linear independence assumption $\mathbf{v}$ is a vertex of $\mathrm{C}_i^{(n)}$. ∎

Lemma 3.3. *For any $\mathbf{v} \in \mathrm{V}^{(n)}$ and for any $i > 0$, $\mathbf{v}$ is a vertex of $\mathrm{C}_i^{(n)}$ iff $i \in \mathrm{I}^{(n)}(\mathbf{v})$.*

Proof. If $\mathbf{v}$ is a vertex of $C_i^{(n)}$, then $h_i(\mathbf{v}) = B^{(n)}(\mathbf{v})$ so $i \in I^{(n)}(\mathbf{v})$. Conversely, if $i \in I^{(n)}(\mathbf{v})$ then $h_i(\mathbf{v}) = B^{(n)}(\mathbf{v})$ so $\mathbf{v} \in C_i^{(n)}$ and by Lemma 3.2, $\mathbf{v}$ is a vertex of $C_i^{(n)}$. ∎

Definition 3.3 (adjacent, adjacency list). *Let $\mathbf{v}_j, \mathbf{v}_k \in V^{(n)}$. Then $\mathbf{v}_j$ and $\mathbf{v}_k$ are adjacent or neighbors if they are on the same edge of some polytope $C_i^{(n)}$, that is, if for some i, there are $m-1$ linearly independent constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$ that are satisfied by both $\mathbf{v}_j$ and $\mathbf{v}_k$. Let*

$$Adj^{(n)}(\mathbf{v}_k) = \{j : \mathbf{v}_j \text{ is adjacent to } \mathbf{v}_k\}.$$

Lemma 3.4. *Two vertices $\mathbf{v}_j, \mathbf{v}_k \in V^{(n)}$ are adjacent iff $I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k)$ contains exactly m elements.*

Proof. If $\mathbf{v}_j$ and $\mathbf{v}_k$ are adjacent, there are $m-1$ linearly independent constraints from the collection $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$ that are satisfied by both $\mathbf{v}_j$ and $\mathbf{v}_k$. These constraints give m indices that must be in *both* $I^{(n)}(\mathbf{v}_j)$ and $I^{(n)}(\mathbf{v}_k)$. Conversely, note that $I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k)$ must contain some $i > 0$ or the two vertices would satisfy the same m linearly independent boundary constraints, and hence be identical. Then by definition of the index sets we can find $m-1$ constraints from $\mathbf{A}_i^{(n)}\mathbf{x} = \mathbf{b}_i^{(n)}$ which are satisfied by both $\mathbf{v}_j$ and $\mathbf{v}_k$, so they are adjacent. ∎

Lemma 3.5. *If $\mathbf{v}_j, \mathbf{v}_k \in V^{(n)}$ are adjacent vertices, and $\mathbf{v}$ is a vertex for which $I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k) \subset I^{(n)}(\mathbf{v})$ then either $\mathbf{v} = \mathbf{v}_j$ or $\mathbf{v} = \mathbf{v}_k$.*

Proof. Note that reasoning as in the last proof, $I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k)$ must contain some $i > 0$. But then if the vertices were distinct, $C_i^{(n)}$ would have three vertices on the same edge, which is not possible since $C_i^{(n)}$ is a polytope. ∎

Lemma 3.6 *Let $\mathbf{v} \in V^{(n)}$. Then if $\mathbf{v}$ is a boundary vertex, it is adjacent to exactly m vertices and if $\mathbf{v}$ is not a boundary vertex it is adjacent to exactly $m+1$ vertices.*

Proof. First note that there are $m+1$ subsets of size m from $I^{(n)}(\mathbf{v})$, so by Lemma 3.5 there are at most $m+1$ adjacent vertices. If $\mathbf{v}$ is not a boundary vertex, each of these subsets refers to $m-1$ linearly independent constraints which, by continuity, are satisfied by some elements of X, and hence by adjacent vertices. If $\mathbf{v}$ is a boundary vertex, one of the subsets refers to the m linearly independent boundary constraints that are satisfied only by $\mathbf{v}$, but by continuity each of the others are satisfied by some elements of X, and hence by adjacent vertices. ∎

3.2 Finding $\mathbf{x}_{n+1}$

Finding $\mathbf{x}_{n+1}$ means finding the global maximizer of $B^{(n)}$. In the one-dimensional case, the value of $B^{(n)}$ at intersection points could be listed and updated after each iteration. By comparing the values of $B^{(n)}$ at these points we could easily find the global maximizer. The result below shows that a similar method can be used in higher dimensions. To be specific, if we have a record of all the vertices in $V^{(n)}$ and the value of $B^{(n)}$ at each vertex, we can simply search through the list to find the vertex $\mathbf{v}_\star$ for which $B^{(n)}$ is the largest, and set $\mathbf{x}_{n+1} = \mathbf{v}_\star$. In practice it is more efficient to use a heap and update the heap after each function evaluation. This way $\mathbf{v}_\star$ will simply be the vertex at the top of the heap.

Theorem 3.1. *The global max of $B^{(n)}$ can only occur at a vertex $\mathbf{v} \in V^{(n)}$.*

Proof. Since the $C_i^{(n)}$'s partition X and $B^{(n)}(x) = h_i(x)$ for $x \in C_i^{(n)}$, it is sufficient to show that for any i the maximum of h_i over $C_i^{(n)}$ occurs at a vertex of $C_i^{(n)}$. This follows immediately from the fact that h_i is convex and $C_i^{(n)}$ is a polytope. ■

3.3 Updating the Upper Envelope

In the one-dimensional case the upper envelope was updated by removing the intersection point at x_{n+1} and replacing it by the two new intersection points that were created between the new function evaluation and each of its closest neighbors. In higher dimensions the procedure is more complicated because many vertices may need to be removed, and more than two new vertices will be created. Fortunately, these are all easy to find using the results given below.

Definition 3.4 (dead & alive). *A vertex* $v \in V^{(n)}$ *is dead if* $h_{n+1}(v) < B^{(n)}(v)$ *and alive if* $h_{n+1}(v) \geq B^{(n)}(v)$.

Theorem 3.2 (updating $V^{(n)}$). *Let the linear independence assumption hold. Then*

$$V^{(n+1)} = \{v \in V^{(n)} : v \textit{ is alive}\} \cup \{v : v \textit{ is a vertex of } C_{n+1}{}^{(n+1)}\}.$$

Proof. If v is an alive vertex of $V^{(n)}$, $v \in C_j^{(n)}$, then $h_j(v) < h_{n+1}(v)$ by the linear independence assumption. So $I^{(n+1)}(v) = I^{(n)}(v)$, and v is a vertex by Lemma 3.1. Conversely, let $v \in V^{(n+1)}$. If $h_{n+1}(v) < B^{(n)}(v)$ then $B^{(n+1)}(v) = h_{n+1}(v)$ so $n+1 \in I^{(n+1)}(v)$ and by Lemma 3.3, v is a vertex of $C_{n+1}{}^{(n+1)}$. Otherwise, by the linear independence assumption $h_{n+1}(v) > B^{(n)}(v) = B^{(n+1)}(v)$ so $I^{(n+1)}(v) = I^{(n)}(v)$ and so v is an alive vertex of $V^{(n)}$. ■

So to update $V^{(n)}$ to $V^{(n+1)}$ we need to remove the dead vertices and add the vertices of $C_{n+1}{}^{(n+1)}$. First consider the problem of removing the dead vertices.

Definition 3.5 (graph-connected). *A set of vertices* $V = \{v_1, \ldots, v_p\}$ *is graph-connected if for any* $v_{j_1}, v_{j_2} \in V$, *there exist* $v_{i_1}, \ldots, v_{i_p} \in V$ *with* $v_{j_1} = v_{i_1}$ *and* $v_{j_2} = v_{i_p}$ *and such that* v_{i_k} *is adjacent to* $v_{i_{k+1}}$ *for all* $k = 1, \ldots, p-1$.

Theorem 3.3. *The set of dead vertices in* $V^{(n)}$ *is graph-connected.*

Proof. Let v_1, v_2 be dead vertices and consider the line between v_1 and v_2. Since $C_{n+1}{}^{(n+1)}$ is convex, $h_{n+1}(x) < B^{(n)}(x)$ for every point x on this line. Suppose as we move from v_1 to v_2 the line passes through the polytopes $C_{j_1}{}^{(n)}, \ldots, C_{j_p}{}^{(n)}$ respectively. It is sufficient to show that given a dead vertex in $C_{j_k}{}^{(n)}$ we can reach $C_{j_{k+1}}{}^{(n)}$ by moving only between adjacent dead vertices.

Now note that any dead vertices of $C_{j_k}{}^{(n)}$ are graph-connected because they are in the intersection of the polytope $C_{j_k}{}^{(n)}$ with the halfspace $H_{j_k} = \{x : h_{n+1}(x) \leq h_{j_k}(x)\}$. Furthermore, $C_{j_k}{}^{(n)} \cap C_{j_{k+1}}{}^{(n)} \cap H_{j_k}$ is non-empty since the line passes through it. But $C_{j_k}{}^{(n)} \cap C_{j_{k+1}}{}^{(n)}$ is a polytope and the non-empty intersection of a polytope with a halfspace contains a vertex of that polytope, so there must be a dead vertex $v \in C_{j_k}{}^{(n)} \cap C_{j_{k+1}}{}^{(n)}$ which can be reached by moving along the adjacencies of the dead vertices of $C_{j_k}{}^{(n)}$. ■

Theorem 3.3 allows us to find all the dead vertices without having to compute $h_{n+1}(v)$ for each vertex. Specifically, starting from the dead vertex $v_\star = x_{n+1}$, we examine all the adjacent vertices. Any that are dead are noted. Then we examine all the neighbors of these vertices, noting any dead ones that have not been found before. We continue in this way until no new dead vertices can be found adjacent to those we already know. By Theorem 3.3 we have found all the dead vertices. Using this method, the only points at which $h_{n+1}(\mathbf{v})$ is evaluated are the dead vertices and their alive neighbors. The value of h_{n+1} at these points will be saved for later use. (See the corollary to Theorem 3.3 below.)

Now consider the vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$. Theorem 3.4 and its corollary provide an easy way to compute the location of each of these vertices.

Theorem 3.4 (finding the vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$). *Let* $\mathbf{v} \in X$. *Then* $\mathbf{v}$ *is a vertex of* $\mathrm{C}_{n+1}{}^{(n+1)}$ *iff either*

(i) $\mathbf{v} \in \mathrm{V}^{(n)}$ *is a dead boundary vertex*

(ii) there exist adjacent vertices $\mathbf{v}_j, \mathbf{v}_k \in \mathrm{V}^{(n)}$, *where* $\mathbf{v}_j$ *is dead,* $\mathbf{v}_k$ *is alive, and* $\alpha \in [0,1)$ *such that*

$$\mathbf{v} = \alpha \mathbf{v}_j + (1-\alpha)\mathbf{v}_k$$
$$h_{n+1}(\mathbf{v}) = \mathrm{B}^{(n)}(\mathbf{v}).$$

Proof. Suppose $\mathbf{v}$ is a vertex of $\mathrm{C}_{n+1}{}^{(n+1)}$. If $\mathbf{v}$ is a boundary vertex, it has always satisfied the same $m+1$ linearly independent boundary constraints, so it is a vertex of $\mathrm{V}^{(n)}$, and since $h_{n+1}(\mathbf{v}) = \mathrm{B}^{(n+1)}(\mathbf{v})$ it is dead. If $\mathbf{v}$ is not a boundary vertex, note that $\mathbf{v}$ must be on an edge of some polytope $\mathrm{C}_i{}^{(n)}$, and therefore there exist vertices $\mathbf{v}_j$ and $\mathbf{v}_k$ on the same edge of $\mathrm{C}_i{}^{(n)}$ such that

$$\mathbf{v} = \alpha \mathbf{v}_j + (1-\alpha)\mathbf{v}_k \quad \text{for some } \alpha : 0 \le \alpha < 1.$$

Now note that $\{\mathbf{x} : h_{n+1}(\mathbf{x}) \le h_i(\mathbf{x})\}$ is a halfspace and the edge between $\mathbf{v}_j$ and $\mathbf{v}_k$ is a line segment. These intersect at $\mathbf{v}$, so one endpoint of the line segment is in the halfspace, the other is not, and hence one of the adjacent pair is dead, the other alive.

Conversely, suppose (i) holds. Since $\mathbf{v}$ is a boundary vertex in $\mathrm{V}^{(n)}$, it satisfies m linearly independent boundary constraints, so it is still a vertex. Since $\mathbf{v}$ is dead, $h_{n+1}(\mathbf{v}) = \mathrm{B}^{(n+1)}(\mathbf{v})$ so $\mathbf{v}$ must be a vertex of $\mathrm{C}_{n+1}{}^{(n+1)}$. Now suppose (ii) holds. Since $\mathbf{v}_j$ and $\mathbf{v}_k$ are adjacent, they are on the same edge of some polytope $\mathrm{C}_i{}^{(n)}$. But $h_i(\mathbf{v}_j) > h_{n+1}(\mathbf{v}_j)$ since $\mathbf{v}_j$ is dead, and $h_i(\mathbf{v}_k) \le h_{n+1}(\mathbf{v}_k)$ since $\mathbf{v}_k$ is alive, so by continuity of h_{n+1} and h_i, there is a point $\mathbf{v}$ on the line between $\mathbf{v}_j$ and $\mathbf{v}_k$ for which $h_{n+1}(\mathbf{v}) = h_i(\mathbf{v})$. This point is a vertex since it satisfies the $m-1$ constraints satisfied by $\mathbf{v}_j$ and $\mathbf{v}_k$ as well as this additional constraint, which must be linearly independent of the others or $\mathbf{v}_j$ and $\mathbf{v}_k$ would either both be alive or both dead. ∎

Corollary. *Let* $\mathbf{v}$, $\mathbf{v}_j$ *and* $\mathbf{v}_k$ *be as in Theorem 3.4 (ii). Then*

$$\alpha = \frac{h_{n+1}(\mathbf{v}_k) - \mathrm{B}^{(n)}(\mathbf{v}_k)}{(h_{n+1}(\mathbf{v}_k) - \mathrm{B}^{(n)}(\mathbf{v}_k)) - (h_{n+1}(\mathbf{v}_j) - \mathrm{B}^{(n)}(\mathbf{v}_j))}.$$

Proof. Since $\mathbf{v}_j$, $\mathbf{v}_k$, and $\mathbf{v}$ are on the same edge of some polytope $\mathrm{C}_i{}^{(n)}$, $h_i(\mathbf{v}_j) = \mathrm{B}^{(n)}(\mathbf{v}_j)$, $h_i(\mathbf{v}_k) = \mathrm{B}^{(n)}(\mathbf{v}_k)$ and $h_i(\mathbf{v}) = \mathrm{B}^{(n)}(\mathbf{v})$. But by the theorem, $h_{n+1}(\mathbf{v}) = \mathrm{B}^{(n)}(\mathbf{v})$ so we have $h_i(\mathbf{v}) = h_{n+1}(\mathbf{v})$. Substituting for $\mathbf{v}$ in terms of $\mathbf{v}_j$ and $\mathbf{v}_k$ and simplifying gives the result. ∎

Theorem 3.4 and its corollary allow us to find the vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$ easily because each dead boundary vertex and each adjacent dead and alive pair were located when finding the dead vertices. The values of h_{n+1} at all these vertices were saved, so all the quantities required for calculating α are readily available.

We also need to find the values of $B^{(n+1)}(\mathbf{v})$ for each $\mathbf{v} \in V^{(n+1)}$. For those that are alive vertices from $V^{(n)}$ this will be just $B^{(n)}(\mathbf{v})$. For those that are dead boundary vertices, we already have $B^{(n+1)}(\mathbf{v}) = h_{n+1}(\mathbf{v})$ since it was saved from before. So the only computations are for the new vertices of ${C_{n+1}}^{(n+1)}$.

Next consider the task of updating the index sets. The index sets of the alive vertices will not change. The index sets of the vertices of ${C_{n+1}}^{(n+1)}$ are easy to work out.

Proposition 3.2. (updating index sets) *Let* $\mathbf{v} \in V^{(n+1)}$.

(i) If $\mathbf{v} \in V^{(n)}$ *and* $\mathbf{v}$ *is alive then* $I^{(n+1)}(\mathbf{v}) = I^{(n)}(\mathbf{v})$.

(ii) If $\mathbf{v}$ *is a new vertex formed between* $\mathbf{v}_j$ *and* $\mathbf{v}_k$ *(see Theorem 3.4) then* $I^{(n+1)}(\mathbf{v}) = (I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k)) \cup \{n+1\}$.

(iii) If $\mathbf{v}$ *is a boundary vertex of* ${C_{n+1}}^{(n+1)}$ *then* $I^{(n+1)}(\mathbf{v})$ *can be obtained from* $I^{(n)}(\mathbf{v})$ *by replacing the only positive index by the value* $n+1$.

Proof.

(i) If $\mathbf{v}$ is an alive vertex of $V^{(n)}$, $h_{n+1}(\mathbf{v}) \geq B^{(n)}(\mathbf{v})$ so $I^{(n)}(\mathbf{v}) \subset I^{(n+1)}(\mathbf{v})$, and by Lemma 3.1 they are identical.

(ii) Since $\mathbf{v}$ is between $\mathbf{v}_j$ and $\mathbf{v}_k$ and the constraints are linear, any element of $I^{(n)}(\mathbf{v}_j) \cap I^{(n)}(\mathbf{v}_k)$ will be in $I^{(n+1)}(\mathbf{v})$. Since $B^{(n+1)}(\mathbf{v}) = h_{n+1}(\mathbf{v})$, $n+1$ will also be in $I^{(n+1)}(\mathbf{v})$. This gives a total of $m+1$ elements so the result holds by Lemma 3.1.

(iii) Let $\mathbf{v}$ be a boundary vertex of ${C_{n+1}}^{(n+1)}$. The m negative elements of $I^{(n)}(\mathbf{v})$ never change, so they will be in $I^{(n+1)}(\mathbf{v})$. Now $B^{(n+1)}(\mathbf{v}) = h_{n+1}(\mathbf{v})$ so $n+1 \in I^{(n+1)}(\mathbf{v})$ which gives the required $m+1$ elements. ■

We finally turn to the problem of determining adjacencies.

Proposition 3.3 (updating adjacencies). *Let* $\mathbf{v} \in V^{(n+1)}$.

(i) If $\mathbf{v}$ *is an alive vertex from* $V^{(n)}$ *then* $Adj^{(n+1)}(\mathbf{v})$ *can be obtained from* $Adj^{(n)}(\mathbf{v})$ *by replacing any index referring to a dead vertex by the index for the resulting new vertex (Theorem 3.4).*

(ii) If $\mathbf{v}$ *is a new vertex formed between* $\mathbf{v}_j$ *and* $\mathbf{v}_k$ *(see Theorem 3.4) then* $\mathbf{v}$ *is adjacent only to* $\mathbf{v}_k$ *and to other vertices of* ${C_{n+1}}^{(n+1)}$.

(iii) If $\mathbf{v}$ *is a boundary vertex of* ${C_{n+1}}^{(n+1)}$ *then* $\mathbf{v}$ *is adjacent only to other vertices of* ${C_{n+1}}^{(n+1)}$.

Proof.

(i) $I^{(n+1)}(\mathbf{v}) = I^{(n)}(\mathbf{v})$ by Proposition 3.2. The index set of any alive adjacent vertex will also have stayed the same, so the vertex will still be adjacent. Any dead adjacent vertex will have given rise to some new vertex, which by Proposition 3.2 shares m of the indices of $I^{(n)}(\mathbf{v}) = I^{(n+1)}(\mathbf{v})$, so is adjacent to $\mathbf{v}$.

(ii) By (i), the vertex must be adjacent to the alive vertex from which it was created. Now $n+1 \in I^{(n+1)}(\mathbf{v})$ by Proposition 3.2, so the only m-element subset of $I^{(n+1)}(\mathbf{v})$ that does not contain $n+1$ is the subset shared by the alive vertex. All other subsets contain $n+1$, so all other adjacent vertices are vertices of ${C_{n+1}}^{(n+1)}$.

(iii) Suppose $\mathbf{v}$ is adjacent to $\mathbf{v}_i$, so they are on the same edge of some polytope. By Proposition 3.2, $n+1$ is the only positive element of $\mathrm{I}^{(n+1)}(\mathbf{v})$, so they must both be vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$.

Proposition 3.4. (adjacencies of vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$). *Let $\mathbf{v}_1$, $\mathbf{v}_2$ be distinct vertices of $\mathrm{C}_{n+1}{}^{(n+1)}$. If $\mathbf{v}_1$ is a boundary vertex, set $\mathbf{v}_{j_1} = \mathbf{v}_1$. Otherwise, let $\mathbf{v}_{j_1}$ be the dead vertex which gave rise to $\mathbf{v}_1$ (see Theorem 3.4). Let d_1 be the index which is in $\mathrm{I}^{(n)}(\mathbf{v}_{j_1})$ but not in $\mathrm{I}^{(n+1)}(\mathbf{v}_1)$. Similarly define $\mathbf{v}_{j_2}$ and d_2. Then the following hold.*

(i) If $\mathbf{v}_{j_1} = \mathbf{v}_{j_2}$ then $\mathbf{v}_1$ and $\mathbf{v}_2$ are adjacent.

(ii) If $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ are adjacent, $\mathbf{v}_1$ and $\mathbf{v}_2$ are adjacent iff $d_1 = d_2$.

(iii) If $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) \cap \mathrm{I}^{(n)}(\mathbf{v}_{j_2})$ contains $m-1$ elements, $\mathbf{v}_1$ and $\mathbf{v}_2$ are adjacent iff $d_1 \notin \mathrm{I}^{(n)}(\mathbf{v}_{j_2})$ and $d_2 \notin \mathrm{I}^{(n)}(\mathbf{v}_{j_1})$.

(iv) If $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) \cap \mathrm{I}^{(n)}(\mathbf{v}_{j_2})$ contains fewer than $m-1$ elements, $\mathbf{v}_1$ and $\mathbf{v}_2$ are not adjacent.

Proof.

(i) Let $\mathbf{v}_{j_1} = \mathbf{v}_{j_2} = \mathbf{v}_j$. Since $\mathrm{I}^{(n+1)}(\mathbf{v}_1)$ contains every element of $\mathrm{I}^{(n)}(\mathbf{v}_j)$ except d_1 and $\mathrm{I}^{(n+1)}(\mathbf{v}_2)$ contains every element but d_2, each must contain the same $m-1$ element subset. But they also contain the element $n+1$, giving m indices in common, so $\mathbf{v}_1$ and $\mathbf{v}_2$ are adjacent.

(ii) Since $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ are adjacent, we can write $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) = \{b_1, a_1, a_2, \ldots, a_m\}$ and $\mathrm{I}^{(n)}(\mathbf{v}_{j_2}) = \{b_2, a_1, a_2, \ldots, a_m\}$, with $b_1 \neq b_2$. First note that no new vertex can occur on the edge between $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ since both of these vertices are dead, and clearly no boundary vertex can occur on an edge. So $b_1 \neq d_1$ and $b_2 \neq d_2$. Say $d_1 = a_1$. Then if $d_2 = d_2$, $\mathrm{I}^{(n+1)}(\mathbf{v}_1) \cap \mathrm{I}^{(n+1)}(\mathbf{v}_2) = \{n+1, a_2, \ldots, a_m\}$ so $\mathbf{v}_1$ and $\mathbf{v}_2$ are adjacent, and if $d_2 \neq d_1$, the intersection contains one less element, so $\mathbf{v}_1$ and $\mathbf{v}_2$ are not adjacent.

(iii) Let $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) = \{c_1, b_1, a_1, a_2, \ldots, a_{m-1}\}$ and $\mathrm{I}^{(n)}(\mathbf{v}_{j_2}) = \{c_2, b_2, a_1, a_2, \ldots, a_{m-1}\}$ with $\{c_1, b_1\} \cap \{c_2, b_2\} = \emptyset$. Now if d_1 is one of the a_i's, $\mathrm{I}^{(n+1)}(\mathbf{v}_1) \cap \mathrm{I}^{(n+1)}(\mathbf{v}_2)$ will contain at most $m-1$ elements, namely the other a_i's and $n+1$. Similarly for d_2. So $\mathbf{v}_1$ and $\mathbf{v}_2$ can only be adjacent if neither d_1 nor d_2 are in $\{a_1, a_2, \ldots, a_{m-1}\}$. Conversely, if $d_1 \in \{c_1, b_1\}$ and $d_2 \in \{c_2, b_2\}$, $\mathrm{I}^{(n+1)}(\mathbf{v}_{i_1}) \cap \mathrm{I}^{(n+1)}(\mathbf{v}_{i_2}) = \{n+1, a_1, a_2, \ldots, a_{m-1}\}$, which has m elements, so $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_2}$ are adjacent.

(iv) If $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) \cap \mathrm{I}^{(n)}(\mathbf{v}_{j_2})$ contains fewer than $m-1$ elements, then $\mathrm{I}^{(n+1)}(\mathbf{v}_1) \cap \mathrm{I}^{(n+1)}(\mathbf{v}_2)$ contains at most one more element, namely $n+1$, so $\mathbf{v}_1$ and $\mathbf{v}_2$ cannot be adjacent. ∎

This result allows us to update the index sets easily provided we save the appropriate index d_i for each new vertex $\mathbf{v}_i$. To make the task easier, the elements of each index set are stored in increasing order.

4. Detailed Description of The Algorithm

The algorithm developed in section 3 is now presented in more detail. Programming details (e.g. organization of the heap, management of arrays to allow new vertices to occupy the places of dead vertices, etc.) are suppressed in the following description.

The parameters m, *maxit*, K, ϵ_1, ϵ_2, $\mathbf{x}_1$, and $\mathbf{a}_j, \mathbf{b}_j$ for $j = 1, \ldots, nc$ (where nc is the number of boundary constraints) must be specified. The user must also supply the vertices of the domain X, and the index sets of these vertices, with each index set in increasing order, the last element equal to 1. Let nv be the number of boundary vertices.

It is necessary to store $\mathbf{x}_\star$ (the current estimated position of the global maximum), $\underline{f}$, $\bar{f}$ and f_{min}, and for each vertex, the values of $\mathbf{v}_j$, $\mathrm{B}^{(n)}(\mathbf{v}_j)$, $\mathrm{I}^{(n)}(\mathbf{v}_j)$ and $Adj(\mathbf{v}_j)$.

1. **Initialize**

   (i) Calculate $f(\mathbf{x}_1), \nabla f(\mathbf{x}_1)$.

   (ii) For every $k = 1, \ldots, nv$ let

   $$\mathrm{B}^{(1)}(\mathbf{v}_k) = h_1(\mathbf{v}_k) = f(\mathbf{x}_1) + \nabla f(\mathbf{x}_1)'(\mathbf{v}_k - \mathbf{x}_1) + K\| \mathbf{v}_k - \mathbf{x}_1 \|^2.$$

   (iii) For every $k = 1, \ldots, nv$, put k in $Adj(\mathbf{v}_k)$. For every $k, j = 1, \ldots, nv$ with $k < j$, if $\mathrm{I}^{(1)}(\mathbf{v}_j) \cap \mathrm{I}^{(1)}(\mathbf{v}_k)$ has m elements, put j in $Adj(\mathbf{v}_k)$ and k in $Adj(\mathbf{v}_j)$.

   (iv) Create a heap of the vertices.

   (v) Let $\mathbf{x}_\star = \mathbf{x}_1$, $\underline{f} = f_1$ and $f_{min} = f_1$.

   (vi) Let $\mathbf{x}_2 = \mathbf{v}_\star$, the vertex from the top of the heap and $\bar{f} = \mathrm{B}^{(1)}(\mathbf{v}_\star)$.

   (vii) Let $n = 1$.

2. **Evaluate the function**
   Compute $f(\mathbf{x}_{n+1}), \nabla f(\mathbf{x}_{n+1})$.
   If $f(\mathbf{x}_{n+1}) > \underline{f}$ then $\mathbf{x}_\star = \mathbf{x}_{n+1}, \underline{f} = f(\mathbf{x}_{n+1})$
   $f_{\min} = \min(f(\mathbf{x}_{n+1}), f_{\min})$
   If $f(\mathbf{x}_{n+1}) > \bar{f}$ stop (K too small).

3. **Update the upper envelope**

   (i) *(Find dead vertices).* Classify $\mathbf{v}_\star = \mathbf{x}_{n+1}$ as dead and put it in a list of dead vertices to be considered. Calculate $h_{n+1}(\mathbf{v}_\star)$ and save.

   (A) If the list is empty, go to (ii). Otherwise, take a vertex $\mathbf{v}$ off the list.

   (B) Find a vertex $\mathbf{v}_j$ adjacent to $\mathbf{v}$. If $\mathbf{v}_j$ is already classified, go to (C). If $\mathbf{v}_j$ is not yet classified, calculate $h_{n+1}(\mathbf{v}_j)$ and save. If $h_{n+1}(\mathbf{v}_j) < \mathrm{B}^{(n)}(\mathbf{v}_j)$, classify $\mathbf{v}_j$ as dead and add it to the list, otherwise, classify $\mathbf{v}_j$ as alive.

   (C) If all vertices adjacent to $\mathbf{v}$ have been looked at, return to (A), otherwise return to (B).

   (ii) *(New vertices).* For each dead vertex $\mathbf{v}_j$ and adjacent alive vertex $\mathbf{v}_k$ do the following.

   (A) Let $nv = nv + 1$. Let $\mathbf{v}_{nv} = \alpha \mathbf{v}_j + (1 - \alpha)\mathbf{v}_k$, where

   $$\alpha = \frac{h_{n+1}(\mathbf{v}_k) - \mathrm{B}^{(n)}(\mathbf{v}_k)}{(h_{n+1}(\mathbf{v}_k) - \mathrm{B}^{(n)}(\mathbf{v}_k)) - (h_{n+1}(\mathbf{v}_j) - \mathrm{B}^{(n)}(\mathbf{v}_j))}.$$

   (B) Let

   $$\begin{aligned} \mathrm{I}^{(n+1)}(\mathbf{v}_{nv}) &= (\mathrm{I}^{(n)}(\mathbf{v}_j) \cap \mathrm{I}^{(n)}(\mathbf{v}_k)) \cup \{n+1\}, \\ \mathrm{B}^{(n+1)}(\mathbf{v}_{nv}) &= h_{n+1}(\mathbf{v}_{nv}). \end{aligned}$$

(C) Replace j by nv in $Adj(\mathbf{v}_k)$. Let $Adj(\mathbf{v}_{nv}) = \{k\}$.

(D) Update the heap.

(E) Let $d(\mathbf{v}_{nv})$ be the element of $\mathrm{I}^{(n)}(\mathbf{v}_j)$ that was replaced by $n+1$ in step (B). Save to use in part (iv).

(iii) *(Boundary vertices).* For each dead boundary vertex $\mathbf{v}_j$ do the following.

(A) Let $nv = nv + 1$. Let $\mathbf{v}_{nv} = \mathbf{v}_j$.

(B) Obtain $\mathrm{I}^{(n+1)}(\mathbf{v}_{nv})$ from $\mathrm{I}^{(n)}(\mathbf{v}_j)$ by replacing the positive element by $n+1$. Let $\mathrm{B}^{(n+1)}(\mathbf{v}_{nv}) = h_{n+1}(\mathbf{v}_j)$.

(C) Let $Adj(\mathbf{v}_{nv}) = \{nv\}$.

(D) Update the heap.

(E) Let $d(\mathbf{v}_{nv})$ be the element of $\mathrm{I}^{(n)}(\mathbf{v}_j)$ that was replaced by $n+1$ in step (B). Save to use in part (iv).

(iv) *(Update adjacencies).*

(A) For every dead vertex $\mathbf{v}_j$, record that all new vertices obtained from $\mathbf{v}_j$ in part (ii) or (iii) above are adjacent to each other.

(B) For every adjacent pair of dead vertices $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$, look at every pair of new vertices created from these in part (ii) or (iii). Suppose $\mathbf{v}_1$ was created from $\mathbf{v}_{j_1}$ and $\mathbf{v}_2$ was created from $\mathbf{v}_{j_2}$. If $d(\mathbf{v}_1) = d(\mathbf{v}_2)$, then record $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_2}$ as adjacent.

(C) For every adjacent pair of dead vertices $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$, determine whether $\mathrm{I}^{(n)}(\mathbf{v}_{j_1})$ and $\mathrm{I}^{(n)}(\mathbf{v}_{j_2})$ differ by two elements. If so, look at every pair of new vertices created from $\mathbf{v}_{j_1}$ and $\mathbf{v}_{j_2}$ in part (ii) or (iii). Suppose $\mathbf{v}_1$ was created from $\mathbf{v}_{j_1}$ and $\mathbf{v}_2$ was created from $\mathbf{v}_{j_2}$. Then record $\mathbf{v}_1$ and $\mathbf{v}_2$ as adjacent if neither $d(\mathbf{v}_1)$ nor $d(\mathbf{v}_2)$ is in the set $\mathrm{I}^{(n)}(\mathbf{v}_{j_1}) \cap \mathrm{I}^{(n)}(\mathbf{v}_{j_2})$.

4. **Find next point**
Find $\mathbf{x}_{n+2}$ = the vertex from the top of the heap, and let $\bar{f} = \mathrm{B}^{(n+1)}(\mathbf{x}_{n+2})$.

5. **Test for convergence**
If $n = maxit$ or

$$\bar{f} - \underline{f} \le \epsilon_1 \quad \text{and} \quad \frac{\bar{f} - \underline{f}}{\underline{f} - f_{min}} \le \epsilon_2$$

then stop; $\mathbf{x}_\star$ is the estimate of the position of the global maximum and $\underline{f}$ is the function value at this point. Otherwise, set $n = n + 1$ and return to step 2.

5. Computational Behavior

In this section we investigate the storage and cpu time requirements of the algorithm and the dependence of these on the dimensionality of the problem and the number of function evaluations. The example we consider is the cosine example given in the appendix;

$$f(\mathbf{x}) = a \sum_{i=1}^{m} \cos(cx_i) - \sum_{i=1}^{m} x_i^2,$$

with $a = 0.1$, $c = 5\pi$ and $K = 11.34$.

In Dixon & Szegö [9] the authors present the results of several global optimization algorithms for a set of test functions for global optimization. They report the number of function evaluations required by each algorithm and the cpu time in terms of *standard time*. *Standard time* is defined as the time it takes to evaluate the Shekel(5) function 1000 times. This method of assessing the performance of an algorithm, while not without problems, is widely used as a means of comparing algorithms and it seems reasonable that we present results along similar lines.

For this reason, all timing results will be presented in units of *standard time* as defined above. However, we note here that the definition of *standard time*, while an attempt to provide an objective means of comparing results from different machines, is highly dependent on the use of a floating point accelerator. This is because the floating point accelerator speeds up the evaluation of the Shekel(5) function considerably, but programs which are less floating-point intensive are not speeded up as much. Most of the timing results presented in what follows will be given in *fpa standard time.*

5.1 Dependence of Time on Dimensionality and Number of Function Evaluations

The program has been run for the COS function in 1, 2, 3, 4, 5 and 6 dimensions, and for $n =$ 100, 200, 300 and 400 iterations (=function evaluations). It was also run for 1 iteration to give an idea of the setup time. Note that for these examples $\epsilon_2 = 10^{-6}$ and $K = 10000$ were used since otherwise the algorithm converged in less than 400 iterations for $m = 1$. Table 1 gives the time in *fpa standard time*, and the number of vertices for each of these situations.

For any dimension, the time taken goes up roughly in proportion to n, as does the number of vertices, so the time per vertex does not increase significantly as the number of function evaluations increases. However, for 6 dimensions the amount of memory required was so great that page faults were encountered, which probably accounts in part for the rather sharp increase in the time per vertex. The fact that the time per vertex does not appreciably increase with the number of function evaluations (equivalently with the number of vertices) is an indication that the efforts to get efficient updates of $\mathrm{B}^{(n)}$ have largely succeeded.

On the other hand, comparing across dimensions, the increase in both the number of vertices and the cpu time is roughly exponential. Since it is necessary to store the vertices and various facts pertaining to each vertex, the storage required for the algorithm increases rapidly with dimension. In a 32 megabyte machine, virtual memory was required for 400 function evaluations in 6 dimensions. But 400 function evaluations in 6 dimensions is not usually enough to get close to the global maximum for complicated functions.

Table 1

Cpu-time and Number of Vertices for COS function, different dimensions
n is the number of iterations

Time (*fpa standard time*)							
		dimension (m)					
		1	2	3	4	5	6
	1	.32	.32	.32	.32	.48	.52
	100	.77	2.13	6.61	26.77	140.65	658.06
n	200	1.45	3.61	13.87	64.84	380.32	1900.00
	300	2.00	5.48	21.29	107.42	635.16	4360.32
	400	2.58	7.31	29.40	150.00	894.84	15590.52

Number of vertices							
		dimension (m)					
		1	2	3	4	5	6
	1	2	4	8	16	32	64
	100	101	202	522	1576	4787	14796
n	200	201	402	1103	3454	11121	39766
	300	301	602	1675	5388	17691	67304
	400	401	802	2265	7446	25773	97766

5.2 Rate of Convergence

The rate of convergence of the algorithm was investigated for the COS function in 1, 2, 3 and 4 dimensions. Table 2 gives the results of running the algorithm to convergence for each of these cases with $\epsilon_1 = 10^{-2}$, and $\epsilon_2 = 10^{-4}$. The number of function evaluations and the time increases rapidly with the dimensionality of the function, but the number of local maxima are also increasing. Again, the different co-processors give very different results.

Table 2
Results of running the algorithm to convergence, COS function

dimensions m	number of function evaluations	number of local max	soft std time	68881 std time	fpa std time
1	19	5	.05	.38	.82
2	77	25	.53	1.06	1.83
3	327	125	7.04	13.75	23.66
4	1392	625	179.88	441.75	660.97

5.3 Performance on Test Functions

In this section we look at the performance of the algorithm for a number of simple examples and where possible compare it to published results.

The appendix contains definitions of the example functions. In Table A6 we list various facts about the functions, including the upper bound constant K, the number of local maxima, the number of these that are global, and the location and value of the global maximum. The examples we consider here are the one-dimensional functions WingoA, WingoB and WingoC, the two-dimensional functions EXP2, COS2, S&H, GW, G&P, RCOS and C6, the three-dimensional functions F&N and H3, and the four-dimensional EXP4 and COS4. The four-dimensional examples S5, S7 and S10 and the six-dimensional H6 given in Dixon & Szegö [9] were not considered here because for these functions the algorithm converged slowly and required a large amount of workspace.

In Table 3 we present the results of running the upper bound algorithm. For each example, $\epsilon_1 = 10^{-2}$ and $\epsilon_2 = 10^{-4}$. The time has been given in units of *fpa standard time.* Approximate values for *soft standard time* have been obtained for most functions by dividing the *fpa standard time* by 4. As with all optimization routines, this algorithm works better for some functions than others.

Wingo [27] used a derivative-free algorithm due to Brent [5] to find the global maximum for the three Wingo functions. Brent's method took 71, 107 and 1048 function evaluations for the three functions. The upper bound algorithm took 16, 21 and 391 function evaluations for the same functions, but these should be multiplied by two since with each function evaluation there is a derivative evaluation. We also note that Wingo used a different stopping criterion than we used. Even taking these things into consideration, the upper bound algorithm does seem to compare favorably with Brent's algorithm for these 1-dimensional functions.

Table 4 gives results from the literature in terms of the number of function evaluations and the units of standard time. It must be noted that it is difficult to compare algorithms in this way because the algorithms employ different stopping rules. If, for example, we had selected different values of ϵ_1 and ϵ_2, we could have increased or decreased the number of function evaluations and standard time considerably. Also, for a fair comparison, the number of function evaluations for the upper bound algorithm must be multiplied by $m + 1$ since for each function we perform a gradient evaluation.

In terms of function evaluations, the upper bound algorithm seems to compare reasonably well with the other algorithms for the functions GW, C6 and RCOS. It does not work as well for G&P, for which the algorithm does not converge in 10000 iterations. It is also not particularly good for H3. In terms of time, the upper bound algorithm compares favorably with the other algorithms for the RCOS function but not for any of the other functions.

One problem is in the analytic computation of a good upper bound constant. A more difficult problem is that the upper bound constant may differ drastically over different subregions of X and its maximum value on X is very high, but occurs only in a small subregion not containing the global optimum. This seems to be the problem with the (G & P) function. On the other hand, the upper bound constant for RCOS is "uniformly fairly good" over the entire region.

At this stage, the upper bound algorithm seems rather more expensive than the methods available from the literature. However, all of the algorithms taken from the literature are stochastic and may fail to locate the global maximum a significant proportion of the time, whereas the upper bound algorithm locates it with certainty when the upper bound constant is known. Moreover, not all the researchers have published results on how often the algorithm finds the global maximum, so it is difficult to know how reliable some of these methods are. Another point in favor of the upper bound algorithm is that although the overhead is high, leading to high values for standard time, the number of function evaluations is often competitive with the existing algorithms. If a function is expensive to evaluate the overhead is less significant. Then the upper bound algorithm may do better than many of its competitors.

We have also found (Cutler [6]) that combining the upper bound algorithm with an occasional local search usually results in a much faster convergence and a subsequent reduction in cpu-time. In fact, this combination is competitive with the best timing results for published algorithms. This modified algorithm will be the subject of a forthcoming report.

A RATFOR or FORTRAN listing of the upper bound algorithm is available by writing to A. Cutler, Dept. of Mathematics & Statistics, Lund Hall, Utah State University, Logan, Utah 84322-3900 or is available via e-mail (address: adele@sunfs.math.usu.edu)

Table 3
Results of Upper Bound Algorithm for Example Functions.

Function definitions are given in the appendix. K is the upper bound constant, n is the number of function evaluations, $\mathbf{x}_\star$ is the location of $f_{\max}$, the largest function value obtained in the n evaluations.

function	K	n	$\mathbf{x}_\star$	$f_{\max}$	*fpa* std time	*soft* std time (est.)
WingoA	.5	16	7.068	-15.282	.72	.18
WingoB	1.25	21	7.724	-44.957	.77	.19
WingoC	3.125	391	118.487	-261.787	4.49	1.12
EXP2	.223	24	(-7.3e-4,-3.0e-3)	1.000	.95	.24
EXP4	.223	117	(-4.2e-3,1.6e-3, 9.4e-4,1.4e-3)	1.000	33.48	8.37
COS2	11.34	77	(-1.5e-3,-3.5e-4)	0.200	1.83	.53
COS4	11.34	1392	(9.7e-4,7.5e-4, 1.2e-3,1.1e-3)	0.400	660.97	179.88
S&H	45.35	667	(12.579,1.751)	95.283	19.16	4.79
GW	.495	939	(-3.0e-2,5.0e-2)	1.000	17.65	4.41
G&P	1.7e6	10000^N	(2.7e-3,-1.001)	-3.002	187.81	46.95
RCOS	8.56	269	(9.421,2.489)	-.398	4.92	1.23
C6	4.5	112	(-9.2e-2,.715)	1.032	2.42	.60
F&N	13.11	8657	(110.33,6.88,.89)	-74.196	869.10	217.27
H3	197.1	2575	(.117,.555,.852)	3.863	213.42	53.35

Table 4
Comparison with Other Methods.

Method	Number of function evaluations					Units of standard time				
	Function					Function				
	GW	C6	G&P	RCOS	H3	GW	C6	G&P	RCOS	H3
Törn [24]	-	-	2499	1558	2584	-	-	4	4	8
de Biase & Frontini [1] & Frontini [1]	-	-	378	597	732	-	-	15	14	17
Price [16]	-	-	2500	1600	2400	-	-	3	4	8
Bremermann [4] †	-	-	300	160	420^L	-	-	0.7	0.5	2^L
Rinnooy Kan & Timmer [19&20]	-	-	148	206	197	-	-	0.15	0.25	0.5
Snyman & Fatti [23]	1496	178	474	-	365	1.4	0.1	0.2	-	0.6
Vanderbilt & Louie [25]	-	-	1186	557	1224	-	-	2	1	4
New Method	939	112	N	269	2575	4.41	.6	N	1.23	53.35

† Results taken from Dixon & Szegö [9]
L : the algorithm found a local maximum.
N : The algorithm did not converge in 10000 iterations.

Appendix: Example Functions

Most of the following examples were taken from Dixon & Szegö [9]. We also included a number of statistical examples.

The dimensionality and the location of the global maxima for these functions are summarized in Table A6.

Cauchy Likelihood: WingoA, WingoB, WingoC (Wingo [27]).

- Dimensions: 1.
- Definition: For a given set of data $y_1 \leq y_2 \leq \ldots \leq y_n$, we wish to maximize the log likelihood for the one-parameter Cauchy distribution. This entails maximizing

$$f(x) = -\sum_{i=1}^{n} \left[log(\pi) + log(1 + (y_i - x)^2) \right]$$

  with respect to x. The values of y_i given by Wingo [27] are in Table A1.
- Region of interest: $y_1 \leq x \leq y_n$.
- Starting point: 9.5 for WingoA, 13.0 for WingoB, 242.5 for WingoC.

Exponential: EXP

- Dimensions: variable.
- Definition:

$$f(\mathbf{x}) = e^{-\frac{1}{2}\sum_{i=1}^{m} x_i^2}.$$

- Region of interest: $-1 \leq x_i \leq 1$ for $i = 1, \ldots, m$.
- Starting point: $x_i = 0.2$ for $i = 1, \ldots, m$.

Cosine Mixture: COS

- Dimensions: variable.
- Definition:

$$f(\mathbf{x}) = a\sum_{i=1}^{m} \cos(cx_i) - \sum_{i=1}^{m} {x_i}^2 \qquad a > 0, c > 0.$$

- For our example, let $a = 0.1$ and $c = 5\pi$.
- Region of interest: $-1 \leq x_i \leq 1$ for $i = 1, \ldots, m$.
- Starting point: $x_i = 0.5$ for $i = 1, \ldots, m$.

Poissonian Pulse-train Likelihood: S&H (Slump & Hoenders [22]).

- Dimensions: 2.

- Definition:
$$f(\mathbf{x}) = \sum_{i=1}^{p} (-\lambda_i(\mathbf{x}) + \hat{n}_i log(\lambda_i(\mathbf{x})) - log(\hat{n}_i!)),$$
where
$$\lambda_i(\mathbf{x}) = \alpha \left[1 + \beta\, exp\left\{-\frac{1}{2}\left(\frac{i - x_1}{x_2}\right)^2\right\}\right] + \lambda_0$$
and $\alpha = 2.0, \beta = 2.5, \lambda_0 = 3.0, p = 21$ and the values of $\hat{n}_i, i = 1, \ldots, p$ are given in Table A2.
- Region of interest: $1 \leq x_1 \leq 21, 1 \leq x_2 \leq 8$.
- Starting point: $x_1 = 11.0, x_2 = 4.5$.

Griewank: GW (Griewank [13]).

- Dimensions: 2.
- Definition:
$$f(\mathbf{x}) = -\sum_{i=1}^{m} \frac{x_i^2}{d} + \prod_{i=1}^{m} \cos\left(\frac{x_i}{\sqrt{i}}\right).$$
We consider only $m = 2$ and $d = 200$.
- Region of interest: $-100 \leq x_i \leq 100$ for $i = 1, \ldots, m$.
- Starting point: $x_1 \leq 25.0, x_2 \leq 25.0$.

Goldstein & Price: G&P (Dixon & Szegö [9]).

- Dimensions: 2.
- Definition:
$$\begin{aligned} f(\mathbf{x}) \;&=\; -\left[1 + (x_1 + x_2 + 1)^2({3x_1}^2 + {3x_2}^2 + 6x_1x_2 - 14x_1 - 14x_2 + 19)\right] * \\ &\quad \left[30 + (2x_1 - 3x_2)^2({12x_1}^2 + {27x_2}^2 - 36x_1x_2 - 32x_1 + 48x_2 + 18)\right]. \end{aligned}$$
- Region of interest: $-2 \leq x_1 \leq 2$ and $-2 \leq x_2 \leq 2$.
- Starting point: $x_1 = -1.0, x_2 = 1.0$.

Branin: RCOS (Dixon & Szegö [9]).

- Dimensions: 2.
- Definition:
$$f(\mathbf{x}) = -\left[a(x_2 - b{x_1}^2 + cx_1 - d)^2 + e(1 - f)\cos x_1 + e\right],$$
where $a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, d = 6, e = 10, f = \frac{1}{8\pi}$.
- Region of interest: $-5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$.
- Starting point: $x_1 \leq 0.0, x_2 \leq 5.0$.

Six-Hump Camel Back Function: C6

- Dimensions: 2.
- Definition:

$$f(\mathbf{x}) = -(4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4).$$

- Region of interest: $-5 \le x_1 \le 5, -5 \le x_2 \le 5$.
- Starting point: $x_1 = 0.0, x_2 = 0.0$.

Binomial Likelihood: F&N (Freedman & Navidi [11]).

- Dimensions: 3.
- Definition:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \left[v_i \, log(1 - e^{-f_i}) - f_i(p_i - v_i) \right],$$

  where

$$f_i(\mathbf{x}) = x_1 t_i^6 (1 + x_2 d_i)(1 + x_3 d_i),$$

  and $t_i, d_i, p_i, and\ v_i$ are given in Table A3.

- Region of interest: $100 \le x_1 \le 140, 0.1 \le x_2 \le 8, 0.1 \le x_3 \le 8$, with $x_3 \le x_2$.
- Starting point: $x_1 = 120.0, x_2 = 4.0, x_3 = 2.0$.

Hartman: H3 (Dixon & Szegö [9]).

- Dimensions: 3.
- Definition:

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i e^{-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2},$$

  where the a_{ij}, p_{ij} and c_i are given in Table A4.

- Region of interest: $0 \le x_i \le 1$ for $i = 1, \ldots, 3$.
- Starting point : $x_1 = 0.6, x_2 = 0.7, x_3 = 0.8$.

Shekel: S5 (Dixon & Szegö [9]).

- Dimensions: 4.
- Definition:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \frac{1}{(\mathbf{x} - \mathbf{a}_i)'(\mathbf{x} - \mathbf{a}_i) + \mathbf{c}_i}$$

  where the a_{ij}, c_i are given in Table A5.

- Region of interest: $0 \le x_i \le 10$ for $i = 1, \ldots, m$.
- Starting point: $x_1 = 6.0, x_2 = 7.0, x_3 = 8.0, x_4 = 9.0$.

Table A1
Data for Wingo functions.

Function	Data									
Wingo A	3	7	12	17						
Wingo B	2	5	7	8	11	15	17	21	23	26
Wingo C	4.1	7.7	17.5	31.4	32.7	92.4	115.3	118.3	119.0	129.6
	198.6	200.7	242.5	255.0	274.7	274.7	303.8	334.1	430.0	489.1
	703.4	978.0	1656.0	1697.8	2745.6					

Table A2
Data for S&H function.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$\hat{n}_i$	5	2	4	2	7	2	4	5	4	4	15	10	8	15	5	6	3	4	5	2	6

Table A3
Data for F&N function.

i	t_i	d_i	p_i	v_i	i	t_i	d_i	p_i	v_i
1	.18	.00	401	3	16	.33	.60	11	7
2	.24	.00	383	5	17	.17	.75	134	7
3	.33	.00	23	4	18	.18	.75	267	12
4	.18	.30	1573	25	19	.24	.75	311	69
5	.24	.30	900	83	20	.33	.75	12	9
6	.33	.30	92	35	21	.17	1.00	67	3
7	.17	.35	389	9	22	.18	1.00	131	5
8	.18	.35	792	19	23	.24	1.00	160	44
9	.24	.35	639	52	24	.33	1.00	10	10
10	.33	.35	45	21	25	.15	1.50	90	5
11	.18	.45	383	3	26	.16	1.50	86	6
12	.24	.45	445	39	27	.17	1.50	65	4
13	.33	.45	12	5	28	.18	1.50	121	12
14	.18	.60	268	6	29	.24	1.50	130	50
15	.24	.60	415	60					

Table A4
Data for H3 functions.

i	a_{ij}			c_i	p_{ij}		
1	3.0	10	30	1.0	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3.0	10	30	3.0	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.03815	0.5743	0.8828

Table A5
Data for S5 function.

i	a_{ij}				c_i
1	4.	4.	4.	4.	.1
2	1.	1.	1.	1.	.2
3	8.	8.	8.	8.	.2
4	6.	6.	6.	6.	.4
5	3.	7.	3.	7.	.4

Table A6
Example Functions.

function	m (dims)	#local(global) max	$\mathbf{x}_\star$	$f_\star$	K
WingoA	1	4(1)	7.062	-15.282	.5
WingoB	1	10(1)	7.729	-44.957	1.25
WingoC	1	25(1)	118.497	-261.786	3.125
EXP2	2	1(1)	(0,0)	1	.223
EXP4	4	1(1)	(0,0,0,0)	1	.223
COS2	2	25(1)	(0,0)	.2	11.34
COS4	4	625(1)	(0,0,0,0)	.4	11.34
S&H	2	1(1)	(12.578,1.751)	95.283	45.35
GW	2	713(1)	(0,0)	1	.495
G&P	2	4(1)	(0,-1)	-3	1.7e6
RCOS	2	3(3)	(-3.142,12.275)	-.398	8.56
			(9.425,2.475)	-.398	
			(3.142,2.275)	-.398	
C6	2	6(2)	(8.984e-2,-.713)	1.032	4.5
			(-8.984e-2,.713)	1.032	
F&N	3	1(1)	(109.784,6.989,.874)	-74.194	13.11
H3	3	3(1)	(.115,.556,.853)	3.863	197.1

References

1. de Biase, L. & Frontini, F. (1978). A Stochastic Method for Global Optimization: its Structure and Numerical Performance. *Towards Global Optimization 2, L.C.W. Dixon & G.P. Szegö (eds.), 85–102.* North-Holland Publishing Company (1978).

2. Bowyer, A. (1981). Computing Dirichlet tessellations. *The Computer Journal 24 (1981) 162–166.*

3. Branin, F.H. Jr. & Hoo, S.K. (1972). A Method for Finding Multiple Extrema of a Function of n Variables. *Numerical Methods for Non-linear Optimization, 231–237.* Academic Press.

4. Bremermann, H. (1970). A Method of Unconstrained Global Optimization. *Mathematical Biosciences 9 (1970) 1–15.*

5. Brent, R.P. (1973) *Algoritms for Minimization without Derivatives.* Prentice-Hall.

6. Cutler, A. (1988) *Optimization Methods in Statistics* Ph.D thesis, Univerity of California, Berkeley.

7. Dixon, L.C.W. & Szegö, G.P. (editors). *Towards Global Optimization.* North-Holland / American Elsevier (1975).

8. Dixon, L.C.W. & Szegö, G.P. (editors). *Towards Global Optimization 2.* North-Holland / American Elsevier (1978).

9. Dixon, L.C.W. & Szegö, G.P. (1978) The Global Optimisation Problem: An Introduction. *Towards Global Optimization 2, L.C.W. Dixon & G.P. Szegö (eds.), 1–15.* North-Holland Publishing Company (1978).

10. Evtushenko, Y.G. (1971). Numerical Methods for Finding Global Extrema. *Zh. vychisl. Mat. mat. Fiz. 11 (1971) 1390–1403.*

11. Freedman, D.A. & Navidi, W. (1988). On the Multistage Model for Carcinogenesis. *Technical Report No. 97, Statistics Department, U.C. Berkeley, March, 1988.*

12. Goldstein, A.A. & Price, J.F. (1971). On Descent From Local Minima. *Mathematics of Computation 25(115) (1971) 569–574.*

13. Griewank, A.O. (1981). Generalized Descent for Global Optimization. *Journal of Optimization Theory and Applications 34(1) (1981) 11–39.*

14. Mladineo, R.H. (1986). An Algorithm for Finding the Global Maximum of a Multimodal, Multivariate Function. *Mathematical Programming 34 (1986) 188–200.*

15. Piyavskii, S.A. (1972). An Algorithm for Finding the Absolute Extremum of a Function. *USSR Comp. Math. Math. Phys 12(4) (1972) 57–67.*

16. Price, W.L. (1978). A Controlled Random Search Procedure for Global Optimization. *Towards Global Optimization 2, L.C.W. Dixon & G.P. Szegö (eds.), 71–84.* North-Holland Publishing Company (1978).

17. PRICE, W.L. (1983). Global Optimization by Controlled Random Search. *Journal of Optimization Theory and Applications 40 (1983) 333–348.*

18. RINNOOY KAN, A.H.G. & TIMMER, G.T. (1985). A Stochastic Approach to Global Optimization. *Numerical Optimization 1984, Boggs, P.T., Byrd, R.H. & Schnabel, R.B. (eds.).* SIAM Philadelphia (1985).

19. RINNOOY KAN, A.H.G., & TIMMER, G.T. (1987I). Stochastic Global Optimization Methods. Part I: Clustering Methods. *Mathematical Programming 39 (1987) 27–56.*

20. RINNOOY KAN, A.H.G., & TIMMER, G.T. (1987II). Stochastic Global Optimization Methods. Part II: Multi Level Methods. *Mathematical Programming 39 (1987) 57–78.*

21. SHUBERT, B.O. (1972). A Sequential Method Seeking the Global Maximum of a Function. *SIAM J. Numer. Anal. 9(3) (1972) 379–388.*

22. SLUMP, C.H. & HOENDERS, B.J. (1985). The Determination of the Location of the Global Maximum of a Function in the Presence of Several Local Extrema. *IEEE Transactions on Information Theory IT-31 (1985) 490–497.*

23. SNYMAN, J.A. & FATTI, L.P. (1987). A Multi-Start Global Minimization Algorithm with Dynamic Search Trajectories. *Journal of Optimization Theory and Applications 54 (1987) 121–141.*

24. TÖRN, A.A. (1978). A Search-Clustering Approach to Global Optimization. *Towards Global Optimization 2, L.C.W. Dixon & G.P. Szegö (eds.), 49–62.* North-Holland Publishing Company (1978).

25. VANDERBILT, D. & LOUIE, S.G. (1984). A Monte Carlo Simulated Annealing Approach to Optimization over Continuous Variables. *J. Comput. Phys. 56 (1984) 259–271.*

26. WATSON, D.F. (1981). Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal 24 (1981) 167–172.*

27. WINGO, D.R. (1983). Estimating the Location of the Cauchy Distribution by Numerical Global Optimization. *Commun. Statist. -Simula. Computa. 12(2) (1983) 201–212.*

28. YEMELICHEV, V.A., KOVALEV, M.M., & KRAVTSOV, M.K. (1984). *Polytopes, Graphs and Optimisation.* Cambridge University Press.

TECHNICAL REPORTS

Statistics Department

University of California, Berkeley

1. BREIMAN, L. and FREEDMAN, D. (Nov. 1981, revised Feb. 1982). How many variables should be entered in a regression equation? *Jour. Amer. Statist. Assoc.,* March 1983, **78**, No. 381, 131-136.

2. BRILLINGER, D. R. (Jan. 1982). Some contrasting examples of the time and frequency domain approaches to time series analysis. *Time Series Methods in Hydrosciences,* (A. H. El-Shaarawi and S. R. Esterby, eds.) Elsevier Scientific Publishing Co., Amsterdam, 1982, pp. 1-15.

3. DOKSUM, K. A. (Jan. 1982). On the performance of estimates in proportional hazard and log-linear models. *Survival Analysis,* (John Crowley and Richard A. Johnson, eds.) IMS Lecture Notes - Monograph Series, (Shanti S. Gupta, series ed.) 1982, 74-84.

4. BICKEL, P. J. and BREIMAN, L. (Feb. 1982). Sums of functions of nearest neighbor distances, moment bounds, limit theorems and a goodness of fit test. *Ann. Prob.,* Feb. 1982, **11**. No. 1, 185-214.

5. BRILLINGER, D. R. and TUKEY, J. W. (March 1982). Spectrum estimation and system identification relying on a Fourier transform. *The Collected Works of J. W. Tukey,* vol. 2, Wadsworth, 1985, 1001-1141.

6. BERAN, R. (May 1982). Jackknife approximation to bootstrap estimates. *Ann. Statist.,* March 1984, **12** No. 1, 101-118.

7. BICKEL, P. J. and FREEDMAN, D. A. (June 1982). Bootstrapping regression models with many parameters. *Lehmann Festschrift,* (P. J. Bickel, K. Doksum and J. L. Hodges, Jr., eds.) Wadsworth Press, Belmont, 1983, 28-48.

8. BICKEL, P. J. and COLLINS, J. (March 1982). Minimizing Fisher information over mixtures of distributions. *Sankhyā,* 1983, **45**, Series A, Pt. 1, 1-19.

9. BREIMAN, L. and FRIEDMAN, J. (July 1982). Estimating optimal transformations for multiple regression and correlation.

10. FREEDMAN, D. A. and PETERS, S. (July 1982, revised Aug. 1983). Bootstrapping a regression equation: some empirical results. *JASA,* 1984, **79**, 97-106.

11. EATON, M. L. and FREEDMAN, D. A. (Sept. 1982). A remark on adjusting for covariates in multiple regression.

12. BICKEL, P. J. (April 1982). Minimax estimation of the mean of a mean of a normal distribution subject to doing well at a point. *Recent Advances in Statistics,* Academic Press, 1983.

14. FREEDMAN, D. A., ROTHENBERG, T. and SUTCH, R. (Oct. 1982). A review of a residential energy end use model.

15. BRILLINGER, D. and PREISLER, H. (Nov. 1982). Maximum likelihood estimation in a latent variable problem. *Studies in Econometrics, Time Series, and Multivariate Statistics,* (eds. S. Karlin, T. Amemiya, L. A. Goodman). Academic Press, New York, 1983, pp. 31-65.

16. BICKEL, P. J. (Nov. 1982). Robust regression based on infinitesimal neighborhoods. *Ann. Statist.,* Dec. 1984, **12**, 1349-1368.

17. DRAPER, D. C. (Feb. 1983). Rank-based robust analysis of linear models. I. Exposition and review. *Statistical Science,* 1988, Vol.3 No. 2 239-271.

18. DRAPER, D. C. (Feb 1983). Rank-based robust inference in regression models with several observations per cell.

19. FREEDMAN, D. A. and FIENBERG, S. (Feb. 1983, revised April 1983). Statistics and the scientific method, Comments on and reactions to Freedman, A rejoinder to Fienberg's comments. Springer New York 1985 *Cohort Analysis in Social Research,* (W. M. Mason and S. E. Fienberg, eds.).

20. FREEDMAN, D. A. and PETERS, S. C. (March 1983, revised Jan. 1984). Using the bootstrap to evaluate forecasting equations. *J. of Forecasting.* 1985, Vol. 4, 251-262.

21. FREEDMAN, D. A. and PETERS, S. C. (March 1983, revised Aug. 1983). Bootstrapping an econometric model: some empirical results. *JBES,* 1985, 2, 150-158.

22. FREEDMAN, D. A. (March 1983). Structural-equation models: a case study.

23. DAGGETT, R. S. and FREEDMAN, D. (April 1983, revised Sept. 1983). Econometrics and the law: a case study in the proof of antitrust damages. *Proc. of the Berkeley Conference,* in honor of Jerzy Neyman and Jack Kiefer. Vol I pp. 123-172. (L. Le Cam, R. Olshen eds.) Wadsworth, 1985.