A New Branch-and-Bound Method for Global Optimization

Kaj Madsen and Serguei Zertchaninov*

Abstract The Interval Branch-and-Bound Method for global optimization over a compact right parallelepiped, parallel to the coordinate axes, forms the basis of a new stochastic method which can be applied even when function values are only known as black boxes. The stochastic method is described and convergence to the set of global minimizers is proved under the assumption that the number of stationary points is finite. The method has been implemented in C++, and numerical experiments with some well known test problems in up to 10 variables are described. Comparisons are made with the interval method.

1 Introduction

Many technical and economic problems can be formulated as mathematical programming problems, i.e. as the minimization of a function $f : D \to \Re$ where $D \subseteq \Re^n$. Very often the function f has several local minima only one of those being interesting, namely the smallest one:

$$f^* \equiv \inf\{f(x) \mid x \in D\}. \tag{1}$$

The infinum value f^* is called the *global minimum* and the set of points where it is attained is called the set of *global minimizers*.

The problem of constructing an automatic procedure (i.e. a computer program) which is guaranteed to find this set has not been solved, and it probably never will be. Interval analytic methods solve the problem for a limited class of problems, as mentioned below. Most other methods either give no guarantee for convergence, or convergence is in general (i.e. for larger dimensions) prohibitively slow. Some techniques are semi-automatic, i.e. it is required that a skilled engineer uses his insight into the technical problem which is modeled by the optimization problem to 'help' the optimization method. Often this can lead to solutions which could not have been found by fully automatic optimization methods. An example of such a technique is given in [1].

In this paper we develop an automatic optimization method. Having the exponential complexity of the global optimization problem in mind, we do not intend to guarantee that the solution is found, but rather to try to get the best possible result that can be obtained by using a limited amount of computational resources. Thus the user should

*on leave from Informatics Department, Nizhnij Novgorod State University

specify how much storage and computer time can be afforded rather than specifying some accuracy tolerance as reqired by many traditional approaches.
Since the classical deterministic method based on interval arithmetic has been rather successful (see e.g. the book of Eldon Hansen, [6]) we base our new technique on the same general branch-and-bound principle, but without the requirement that interval arithmetic must be applicable. Thus the new method is a strategy for managing local methods (as e.g. quasi-Newton methods) in a branch-and-bound search for global minimizers.
The basic interval strategy is the following: At any stage of the procedure we assume to have a finite number of

subdomains $D_i \subseteq D$
lower bounds, $LB(D_i)$, on $min\{f(x) \mid x \in D_i\}$
upper bounds, $UB(D_i)$, on $max\{f(x) \mid x \in D_i\}$

with the property that any global minimizer is contained in the union of all D_i.
Throughout the paper we assume that D is a compact right parallelepiped parallel to the coordinate axes (denoted a *box*). This assumption could be weakened but it does not change the basic ideas, and it makes the techniques much simpler. Furthermore we assume that f is a smooth function (i.e. twice continuously differentiable). This assumption is necessary to ensure that the local optimization methods will work.
In Section 2 the general branch-and-bound scheme is given, and two realizations of the scheme are described:

(a) The interval method which has guaranteed convergence to the set of global minimizers. Drawback: It is required that an explicit expression for calculating function values $f(x)$ can be supplied.

(b) A new stochastic strategy intended to estimate the interval method without having the need of being rigorous. This method is generally applicable (i.e. finding $f(x)$ may be considered a black box calculation). Drawback: The guaranteed convergence property is lost.

In Section 3 the new algorithm, based on (b), is presented. Section 4 describes numerical experiments and comparisons with other methods.

2 The general Branch-and-Bound Scheme

We wish to find the global minimum f^* as well as all points x^* for which $f(x^*) = f^*$. Using the notation

$$f(X) \equiv \{f(x) \mid x \in X\}, \qquad X \subseteq D \tag{2}$$

initialize $\rightarrow C, fbound$
while not *stop* **do**
  remove-best-box$(C) \rightarrow B$
  reduce-or-subdivide$(B) \rightarrow result, fbound$
  $C = C \cup \{result\}$
  for $D_i \in C$ **do if** $LB(D_i) > fbound$ **then** $C = C \setminus D_i$
end

Figure 1: Model algorithm

for the range of f over X the problem can be formulated as follows,

$$\begin{array}{ll} \text{find} & f^* = \inf\{f(D)\} \\ \text{and} & X^* = \{x \in D \mid f(x) = f^*\} \end{array} \tag{3}$$

Futhermore, we use the notation C for a finite set of subdomains $D_i, i = 1, ..., p,$ of D with the properties

$$\begin{array}{c} D_i \text{ is a box} \\ X^* \subseteq \cup_{i=1}^{p} D_i \subseteq D \end{array} \tag{4}$$

C is denoted *the candidate set* and the Branch-and-Bound Scheme aims to reduce C and make it converge to X^*. In our description we shall use the notations $p_i \equiv D_i$ if D_i is a *point* (a degenerate box) and $B_i \equiv D_i$ otherwise (i.e. D_i is a *Box*), so $C = \{D_i\} = \{p_i\} \cup \{B_i\}$.
Initially $C = \{D\} = \{B_1\}$ assuming the problem is non-trivial. Now the idea is to consider the "smallest" (in some sense related to the function values) non-trivial element B_i from C and try to reduce it without violating (4) . If no reduction takes place, however, then B_i is subdivided into two or more non-trivial pieces. In any case B_i is removed from C, and either the reduced set or the pieces (denoted *result* below) is added to C. An important feature is the dynamic threshold value, *fbound* say, which is used to reduce C: *fbound* is an upper limit for f^* so if for some value of i $LB(D_i) > fbound$ then D_i can be removed from C. Therefore the global minimum is included as follows

$$min\{LB(D_i) \mid D_i \in C\} \leq f^* \leq fbound \tag{5}$$

The model algorithm is shown in Figure 1. Of course the **for**-statement needs not always search the whole of C. This depends on the data structure used. Futhermore, it only needs to be executed when *fbound* has been changed. *stop* is a function which returns true when all elements of C are points or the inclusion (5) of the minimum is sufficiently narrow. Since any known function value can be used as *fbound* ($f^* \leq f(x)$ for any $x \in D$) *fbound* is reduced every time a better function value is found.
This basic algorithm was first published by Stig Skelboe [15] who used it in an interval analytic implementation for minimizing rational functions by bisection of subdomains.

2.1 Interval Version

Interval arithmetic and analysis was introduced during the 1960's and the standard introductory textbook was authored by Ramon E. Moore, [10]. In interval arithmetic the basic arithmetic operations $\{+, -, *, /\}$ on real numbers are replaced by corresponding operations on real intervals. Furthermore the standard functions $(sin(.), cos(.), exp(.)$, etc.) are defined on the set of real intervals. Therefore, if a real function value $f(b)$, with $b \in \Re^n$, can be calculated by a finite number of these operations and standard functions then it is straight forward to insert intervals instead of real numbers in the calculation. This provides an *interval function* value $F(B)$ (where B is a box) with the properties

$$\begin{aligned} f(B) &\subseteq F(B) \\ d(f(B), F(B)) &= O(w(B)) \end{aligned} \tag{6}$$

where $w(B)$ denotes the *width* of B,

$$\begin{aligned} w(B) &= \sum_{i=1}^{n}(\overline{b_i} - \underline{b_i}) \\ B &= \{b \mid \underline{b_i} \le b_i \le \overline{b_i}\} \end{aligned} \tag{7}$$

and d is the distance between two intervals, i.e. the maximum of the distances between corresponding end points. Thus interval arithmetic provides automatic calculation of lower and upper bounds on function ranges, and if the width of B goes to zero then the overestimate disappears at a linear rate. However, for large values of $w(B)$ the overestimate can be quite severe.
The interval global optimization method was basically developed during the 1970's ([15],[11],[4],[5]) but since then many researchers have improved the method. A thorough description including theoretical as well as practical aspects can be found in [6]. The basic principle is that lower and upper bounds of function ranges are deducted from the interval function values, e.g. $LB(B)$ is defined as the lower bound of $F(B)$, and furthermore $w(B)$ is forced to tend towards zero, if necessary.
The general scheme of Figure 1 is realized as follows:

- *initialize:* $C = D$, $fbound = \max\{F(D)\}$.
- *remove-best-box(C):* The most common strategy is to remove that box B from C which has the smallest value of $\min\{F(B)\}$.
- *reduce-or-subdivide(B):* It is first tested if *monotonicity* of f in B can be detected in some coordinate direction, based on interval extensions of the partial derivatives of f. If monotonicity cannot be detected then it is tested if an interval *Newton-reduction* of f can be made: That is, testing if the interval Newton method, applied to the gradient of f, will reduce B (and hence converge to a stationary point of f in B (see e.g. [10])). One condition for feasibility of this reduction is that there is at most one stationary point in B.
  If both tests fail then B is subdivided into two non-trivial boxes. The most common strategy is to halve that side of B which has the largest width. A thorough

if *monotone* **then** *result*:= mon(B)
else **if** *Newton-reduction* **then** *result*:= Newton(B)
  else *subdivide(B)* $\rightarrow B^1, B^2$
    result:= $\{B^1, B^2\}$

Figure 2: reduce-or-subdivide

discussion of splitting strategies may be found in [3].
Figure 2 shows the structure of the reduction scheme. Here mon(B) is the result of the monotonicity reduction and Newton(B) is the result of the interval Newton iteration: mon(B) $\subseteq B \cap \partial(D)$ (where $\partial(D)$ is the boundary of D), and Newton(B) is either a single stationary point (in case of convergence) or $B \cap \partial(D)$.
fbound is decreased if any smaller function value or upper bound is found during the reduction of B.

2.2 A Stochastic Version

A stochastic realization of the branch-and-bound scheme of Figure 1 was presented in [8]. The motivation is the success of the interval realization of the algorithm which is very efficient when it is applicable, provided the problem dimension n is relatively small. In the stochastic version function and gradient calculations are considered as black boxes - in contrast to the interval version where explicit expressions are needed. The price to be paid is that we loose the safety of the interval method: Lower and upper bounds of function ranges cannot be calculated exactly, they must be estimated.

Therefore the realization is based on information about f (and perhaps f'), calculated at a number of points in the box B. $LB(B)$ is estimated from these points using the known function values and and an estimated Lipschitz constant.
The general scheme of Figure 1 is realized as follows:

- *initialize:* $C = \{D\}$, $fbound = \infty$.
- *remove-best-box(C):* The box B from C which has the smallest (known) function value is chosen.
- *reduce-or-subdivide(B):* The structure of Figure 2 is used.
  The *monotonicity* test of the interval method is replaced by a test based on gradient calculations (or perhaps finite difference estimates to gradients) at a number of randomly chosen points in B. The random test points may be supplemented by some strategically chosen points. If there exists at least one coordinate x_i for which $\partial f / \partial x_i(x)$ has constant sign for all points x in consideration, then we decide that no stationary point exists in B. mon(B) is the result of the monotonicity reduction and is defined as in the interval method.
  The *Newton-reduction* is based on a number of Quasi-Newton iterations started at some of the "best" known points in B. *Newton-reduction* is true if and only if one

of the following two situations occur:
1. If all iteration sequences go out of B then it is decided that B contains no stationary point, and Newton(B) is defined as in the interval version.
2. If all iteration sequences converge to the same point $x \in B$ then we decide that B has exactly one stationary point, and therefore B is discarded and $p = \{x\}$ is added to *result*.
$fbound$ always represents the smallest function value which is seen during the whole calculation.

In this algorithm several parameters have to be chosen: It must be decided how many points are used for the monotonicity test and how many starting points are used for the Newton-reduction. Such points are denoted *sample points* in the following. The fewer sample points the more efficient is the method when applied to "easy" problems. On the other hand, the safety of the method is improved as the number of sample points is increased: The propability of making wrong decisions during the reduction phase becomes smaller. Thus the choice of ideal parameters will depend on an a priori knowledge about the problem which may often be difficult for the user to obtain.

3 The New Method

This Section describes the version of the stochastic strategy of Subsection 2.2 which has produced the results presented in Section 4. Since solutions may be lost because of the uncertainty of the decisions made during the monotonicity and Newton tests the method is imbedded into a loop which restarts the calculation. The idea is to utilize the experience obtained during previous iterations: For instance the subdivision of D and the stationary points which have been obtained may reflect some structure of the objective function, and the basis for choosing the parameters used by the algorithm may have been improved. Therefore we keep an *outer* candidate set G, in addition to the candidate set C, which always includes the whole of D. Thus boxes are never really eliminated during the reduce-and-subdivide phase, but if *result* only covers a subset of the current box B then B (supplied with the information we have found about stationary points in B) is added to a garbage collector G (the outer candidate set). In the outer iteration the garbage collector forms the basis for a re-definition of the inner candidate set C: For each box B in the garbage collector we use the available information about the local mimizers of B in a device which splits B into several boxes that are all added to C. Furthermore parameters which determine the number of points used during the tests of the reduction phase in the inner iteration may be changed in the outer iteration in order to increase the safety of the method. The rationale is that if not all global minimizers are found during the first outer iteration then the problem seems to be harder than expected, and more care should be taken in subsequent iterations.
The structure of the new algorithm is shown in Figure 3. The lower bound reduction is imbedded into the Newton-reduction of the reduce-or-subdivide procedure. We decided to do this because our experiments indicate that it is more efficient only to use the estimated lower bound when there are several stationary points in B.
The boolean *inner-stop* is true when C contains no boxes (i.e. only points). The outer

```
initialize → C, fbound, G
while not time-limit do
    while not inner-stop do
        remove-best-box(C) → B
        reduce-or-subdivide(B) → result, fbound, garbage
        C = C ∪ {result}
        G = G ∪ {garbage}
    end
    SolSet:={p ∈ C | f(p) ≤ fbound + μ}
    while G ≠ ∅ do
        remove(G) → B
        split(B) → result
        C = C ∪ {result}
    end
    perhaps adjust reduction parameters
end
```

Figure 3: New algorithm

stopping condition is a time limit set by the user. The longer time spent, the better result should be expected. For small dimensions the whole set of global minimizers should normally be found in a rather modest amount of computer time. For large dimensions, however, we cannot in general expect to find the solution (3) because of the exponential complexity of the problem, and thus a user determined time limit may be the most reasonable stopping condition.
Many other conditions might be natural: For instance stopping could be based on the relative amount of time spent since the last candidate for a global minimizer was found. One could also include some of the stochastic stopping rules which have been proposed over the years (for a survey, see for instance [2]).
Some details of the algorithm are to be explained.
Consider first the *reduce-or-subdivide* procedure which has a structure very similar to that of Figure 2. The new version is summarized in Figure 4.

- *Sample points.* Two strategies are used for choosing these points. First we use the following regular distribution of up to $2n+1$ points: One point at $m(B)$, the centre of B, and two points for each coordinate direction, $m(B)\pm\frac{1}{3}w(B^{(j)})*e_j, j = 1, ..., n$, where $B = B^{(1)} \times ... \times B^{(n)}$ and e_j is the j-th unit vector. Such points are called *regular*.
  Secondly we may use random points, uniformly distributed in B. These points are called *random.* The total number of sample points, $p_1 = n_{sample}$, has to be provided by the user. The sample points are chosen in the order they are mentioned here (n_{sample} may be less than $2n + 1$).

- *Monotonicity test.* If a stationary point has previously been found in B then f is not likely to be monotone in B, and hence the test is skipped (i.e. *monotone=false*, see Figure 4). Otherwise monotonicity is tested at the sample points as described in Subsection 2.2. If a reduction takes place then *result* is defined as in the interval method: *result*= $B \cap \partial(D)$ in case f decreases towards $\partial(D)$, else *result*=$\emptyset$. The variable *garbage* is set to B.

- *Newton-test.* Local searches for finding stationary points of f in B are started from each sample point. The local technique used in our tests is a version of Powell's Dog-Leg Method, [13], which is efficient and allows for an easy control of the step lengths.
  We distinguish between the following four cases:
  Case 1. Every iteration sequence has an iterate which is outside of B. *result* is defined as in the interval method: *result*$\equiv$ *Newton(B)*=$B\cap\partial(D)$, and *garbage* $= B$. (In practice "outside of B" means "outside of B plus 10 %".)
  Case 2. All iteration sequences provide convergence to the same point $x \in B$. *result*=*Newton(B)*=$\{x\}$ and *garbage* $= B$. (In practice each Newton iteration is stopped by using one of the usual stopping rules for local searches (infinite iterations being prevented), and two stopping points are considered equal if their distance is less than a user provided parameter $p_2 = \epsilon_{cluster}$.)
  Case 3. Several stationary points have been found in B. In this case the lower bound reduction may take place. The lower bound $LB(B)$ is estimated as follows: For every gradient value $f'(x)$ calculated during the Newton-test we find $\|f'(x)\|$ and let *maxgrad* be the maximum of these. Then we estimate $LB(B)$ from the sample points x_i:

$$LB(B) = \min_{i\neq j} \frac{f(x_i) + f(x_j) - maxgrad \cdot \|x_i - x_j\|}{2} \tag{8}$$

  This estimate has been the most successful among several we have tried We have decided not to use all the points generated by the local search in order to perform less calculations. *LB(B)* is corrected if it is greater than the smallest minimum found in the B.
  If $LB(B) > fbound$ then *result* $= \emptyset$ and *garbage* $= B$. Otherwise subdivision takes place.
  Case 4. None of the first 3 situations has occured and subdivision takes place.

- *Subdivision.* If several stationary points have been found in B then the splitting into two is done so each new box contains (known) stationary points. Otherwise the splitting divides B into two equal parts separated by a hyperplane through the centre of B and perpendicular to the side of maximal length. *result* consists of the two new parts and *garbage* $= \emptyset$

The variable $SolSet$ is intended to converge towards the set of global minimizers, X^*. μ is a positive number which is intended to distinguish X^* from other stationary points.

if *monotone* **then**
  result:= $B \cap \partial(D)$
  garbage:= B
else **if** *(case 1)* or *(case 2)* **then**
    result:= Newton(B)
    garbage:= B
  else **if** *(case 3)* and $(LB(B) > fbound)$ **then**
      result:= $\emptyset$
      garbage:= B
    else *subdivide(B)* $\rightarrow B^1, B^2$
      result:= $\{B^1, B^2\}$
      garbage:= $\emptyset$
update *fbound*

Figure 4: New reduce-or-subdivide

In the second inner **while**-loop the garbage set G is emptied. The *remove* procedure takes one box from the garbage set G and stores it in B. Then this box is subdivided into two parts by the *split* procedure.
Different splitting approaches are used depending on the number of stationary points found in B: If there is either none or more than one such points then B is split into two parts as in the *subdivision* above. If there is exactly one stationary point then we find the coordinate for which the distance from this point to the border is maximal. The splitting plane is chosen to be perpendicular to this coordinate axis, and it is shifted from the point to a certain value in the direction where the distance to the boundary is maximal. The distance from the point to the splitting plane is a fixed percentage of the distance from the point to the boundary of B.

We finish this section by analyzing the convergence properties of the method. Under mild conditions we can prove convergence to the set of global minimizers. One requirement is that μ must be chosen sufficiently small, i.e. μ must separate X^* from other stationary points in the sense of inequality (9) below. The key observation in the proof is that no box can remain large during the iteration, i.e. the boxes in the candidate set will all be small when the algorithm has been running for long enough time. (Of course this (finite) time may be enormous and quite unrealistic in practice!)
We need the following definition:

Definition 1 *Let S denote the set of stationary points in D and let $\{x_0, x_1, ...\}$ be the points generated by a search method. The search method is locally stable in D if it has the following properties:*
(i) $\exists \epsilon > 0$: if $s \in S$ and $\|x_0 - s\| < \epsilon$ then $\|x_i - s\| < \epsilon$ for all iterates x_i, and $d(x_i, s) \rightarrow 0$ for $i \rightarrow \infty$.
(ii) $\forall \epsilon > 0 \; \exists \delta > 0$: if $d(x_0, S) > \epsilon$ then there exists an iterate x_i with $\|x_0 - x_i\| > \delta$.

Theorem 1 *Assume the number of stationary points of f is finite, and that the version of the Newton method used is locally stable in D.*
Let $\epsilon > 0$ be given. Then there exists an iteration count k such that after k outer loops
(i) $SolSet \subseteq \{x \mid d(x, X^*) \leq \epsilon\}$
(ii) $\forall x^* \in X^*\ \exists p \in SolSet : d(p, x^*) \leq \epsilon$
provided μ is chosen small enough.

Proof: Consider the first inner while-loop of Figure 3. If *result* is a proper subset of B (see Figure 4) then the dimensionality of *result* is less than that of B, and $garbage = B$. Otherwise the subdivision procedure is used, and since the number of stationary points is finite each of the two new parts must have a volume less than $1 - \eta$*(the volume of B) where $\eta > 0$ is independent of B. Thus for any $\epsilon_1 > 0$ there exists an integer i such that after i inner loops any box treated will have a diameter less than ϵ_1.
Next we show that the first inner while-loop must be finite: If the width of a box B treated in the loop is small enough then the finite number of stationary points and the stability of the search method imply that either monotinicity, case 1 or case 2 will occur during the reduce-and-subdivide phase. Hence B will be reduced to a point (or the dimension of B will be reduced in case of $B \cap \partial(D) \neq \emptyset$) when the volume of B is sufficiently small. Thus each treated box will be either eliminated or reduced to a point p (and hence eliminated from C) when its volume is below a certain limit, and hence the part of D which is covered by the elements of C must disappear after a finite number of iterations. Hence he first inner while-loop must be finite.
Furthermore the elements of G cover D after the first while-loop since $garbage = B$ whenever a reduction of B takes place.
The *split* procedure guarantees that no box in G can remain large during the outer iterations, i.e. for any $\eta > 0$ we obtain $w(B) \leq \eta$ for all boxes $B \in C$ after a certain number of outer iterations.
Let $\epsilon > 0$ be given. Since the search method is locally stable, and since the number of stationary points is finite there exists ϵ_2, $0 < \epsilon_2 \leq \epsilon$, such that $w(B) \leq \epsilon_2$ implies that $B \bigcup S \neq \emptyset$ provides convergence to a stationary point in B, and $B \bigcup S = \emptyset$ provides iteration out of B. Thus $SolSet \subseteq \{x \mid d(x, S) \leq \epsilon_2\} \subseteq \{x \mid d(x, S) \leq \epsilon\}$ after a finite number of outer loops. This proves (i) if $S = X^*$.
Otherwise let

$$\mu = (F^{stat} - F^*)/2 \tag{9}$$

where $F^{stat} = min\{F(s) \mid s \in S \backslash X^*\}$ and $F^* = F(x^*)$, $x^* \in X^*$. Let $\epsilon_3 \leq \epsilon_2$ be chosen so
(a) $d(x, X^*) \leq \epsilon_3 \Rightarrow F(x) < F^* + \mu$
(b) $d(x, S \backslash X^*) \leq \epsilon_3 \Rightarrow F(x) > F^{stat} - \mu$
Now assume $w(B) \leq \epsilon_3$ for all boxes $B \in C$. Then (a) implies

$$F^* \leq fbound < F^* + \mu \tag{10}$$

Therefore (b), the choice of μ, and the definition of $SolSet$ imply $SolSet \subseteq \{x \mid d(x, X^*) \leq \epsilon_3\} \subseteq \{x \mid d(x, X^*) \leq \epsilon\}$.

Finally, let $x^* \in X^*$. If $x^* \in B$ with $w(B) \leq \epsilon_3$ then the first inner loop will provide a point $p \in B$. From the first inequality of (a) and (10) we obtain $p \in SolSet$, and thus (ii) is proved since $d(p, x^*) \leq \epsilon_3 \leq \epsilon$.
This completes the proof. ∎

4 Computational results

The method has been implemented in C++, [16], and the programme is available on request.
In the following we present results from solving 12 test problems of dimensions ranging from 2 to 10. The first 4 function are quite easy for the method since they have only one local (and hence global) minimizer in D. Number 5 has several local and 1 global minimizer in D. Number 6-10 have up to 10^8 local and 1 global minimizer in D. Finally, the last two problems have a few hundred local and 9 global minimizers in D. The problems are listed in Subsection 5.

4.1 Numerical Results

In Table 1 we illustrate capability of the method to find the solution in the case when the cluster radius $p_2 = \epsilon_{cluster}$ is fixed to 0.1, and the number of sample points $p_1 = n_{sample}$ is tuned to provide the best performance. These results are not necessarily the best possible, though, as we have not used random points and have not experimented with p_2.

Table 2 illustrates the situation when the best combination of the parameters is unknown to the user. In this case we let the number of sample points, n_{sample} be $2n+1$, where n is the dimension of the problem, (still having $\epsilon_{cluster} = 0.1$).

Based on our extensive computational experiments with the method we can provide the following general recommendations on the choice of parameters.
If the function to be optimized is unimodal, it is clear that the search sequences, ideally, can lead to no more than one point. In this case just a few (or even one) starting points are usually needed to find the solution (and thus random sampling is not necessary). In case of a multimodal function the number of sample points should be larger. We recommend to use at least n points to start, where n is dimension of the problem. Random sampling sometimes leads to significant speed up.

4.2 Comparison with Multistart

This comparison was carried out to demonstrate the performance gain our method gives compared to an ordinary Multistart method. The principle of the Multistart search is the following: It generates a certain number of random points within the given initial box D, and starts local search sequences from each point. We have constructed a Multistart method, and the results of applying this Multistart method to the test problems are given in Table 3.

Function	Dim.	Local	Global	n_{sample}	First		All	
					F	dF	F	dF
5.1	2	1	1	1	124	104	124	104
5.2	2	1	1	1	15	11	15	11
5.3	3	1	1	1	14	8	14	8
5.4	10	1	1	1	33	13	33	13
5.5	4	10	1	1	30	22	30	22
5.6	4	71000	1	4	999	943	999	943
5.6	5	10^5	1	7	607	585	607	585
5.6	6	10^6	1	5	724	690	724	690
5.6	7	10^8	1	4	574	516	574	516
5.7	10	10^3	1	2	255	197	255	197
5.8	2	400	9	2	17	13	3396	2733
5.9	2	760	9	4	208	199	4494	4350

Where:
Function - test function number
Dim. - dimensionality
Local - number of local minimizers in D
Global - number of global minimizers in D
n_{sample} - number of sample points
First - number of function and derivative calls to find first global minimizer
All - number of function and derivative calls to find all global minimizers

Table 1: Running with tuned parameter n_{sample} and $\epsilon_{cluster} = 0.1$

Function	Dim.	Local	Global	Found	First		All	
					F	dF	F	dF
5.1	2	1	1	1	109	108	109	108
5.2	2	1	1	1	15	11	15	11
5.3	3	1	1	1	14	8	14	8
5.4	10	1	1	1	33	13	33	13
5.5	4	10	1	1	30	22	30	22
5.6	4	71000	1	1	7951	7943	7951	7943
5.6	5	10^5	1	1	950	942	950	942
5.6	6	10^6	1	1	1524	1513	1524	1513
5.6	7	10^8	1	1	2449	2445	2449	2445
5.7	10	10^3	1	1	1196	1178	1196	1178
5.8	2	760	9	9	17	13	10198	10194
5.9	2	400	9	9	192	189	7164	7163

Where:
Function - test function number
Dim. - dimensionality
Local - number of local minimizers in D
Global - number of global minimizers in D
Found - number of found global minimizers
First - number of function and derivative calls to find the first global minimizer
All - number of function and derivative calls to find all global minimizer

Table 2: Running with fixed parameters: $n_{sample} = 2n + 1, \epsilon_{cluster} = 0.1$

Function	Dim.	Exp#1	Exp#2	Exp#3	Exp#4	Exp#5
5.1	2	161	387	27	346	374
5.2	2	10	7	26	8	26
5.3	3	39	32	53	48	34
5.4	10	31	21	22	29	18
5.5	4	340	26	146	343	255
5.6	4	26904	7070	83326	18848	73883
5.6	5	17517	29874	25286	1348	5113
5.6	6	5928	9047	46690	20931	3273
5.6	7	79842	67797	83577	48896	6316
5.7	10	285	312	110	282	218
5.8	2	216 12891	421 10622	538 7375	106 9108	722 10226
5.9	2	66 5133	257 6401	366 7498	447 7497	463 7132

Where:
Function - test function number
Dim. - dimensionality
Exp#n - experiment number n

Table 3: Running Multistart

Function	Dim.	MS first	NM first	MS all	NM all	INT all
5.1	2	259	109			17
5.2	2	16	15			-
5.3	3	42	14			269
5.4	10	25	33			2356
5.5	4	222	30			-
5.6	4	42006	7951			535
5.6	5	15828	950			1486
5.6	6	17174	1524			6137
5.6	7	57286	2445			23795
5.7	10	242	1196			417
5.8	2	401	17	10045	10194	691
5.9	2	320	192	6733	7164	273

Where:
Function - test function number
Dim. - dimensionality
MS first - F(X) evaluations made by Multistart to find first global minimizer
NM first - F(X) evaluations made by new method to find first global minimizer
MS all - F(X) evaluations made by Multistart to find all global minimizers
NM all - F(X) evaluations made by new method to find all global minimizers
INT all - F(X) evaluations made by the Interval Method ([7], version IB) to find all global minimizers ("-" means: Not tested).

Table 4: Comparing with Multistart and Interval Method

Five experiments were made for each test function. The numbers in columns Exp#n indicate the amount of function and gradient calculations needed to find the first global minimizer. For the two problems with several global minimizers there are two lines in the table, the second line providing the the amount of function and gradient calculations needed to find the all minimizers.

In Table 4 we compare mean numbers of function evaluations for successful tests taken from Table 3 with numbers of function evaluations from Table 2. As one can see from the table, for some functions Multistart search outperforms our method. For function 5.4 we should take into consideration the fact that initially 21 points are sampled in the box, and the number of function evaluations for them is included in the numbers from *NM first* and *NM all* columns. We can not state the reason why Multistart outperforms our method for functions 5.8, 5.9 and 5.7. It is possible that the method parameters were chosen unsuccessfully for these tests. For the rest of the functions our method shows significant improvement. The general conclusion is that the more complicated the function, the more our method outperforms Multistart.

5 Test functions

We had to persuade two contrary aims: to select more high-dimensional problems but, at the same time, they should be well known to the majority. The highest dimensionality of our test functions is 10.

Functions with only one local minimizer:

5.1 Rosenbrock function

Dimension: $n = 2$
Interval: $X = [-10, 10]^n$
Formula:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Global minimum: 0
Set of minimizers: $(1, 1)$

5.2 McCormick function

Dimension: $n = 2$
Interval: $X = ([-1.5, 4], [-3, 4])$
Formula:

$$f(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$$

Global minimum: -1.9133
Set of minimizers: $(-0.54719, -1.54719)$

5.3 Box & Betts exponential quadratic sum

Dimension: $n = 3$
Interval: $X = ([0.9, 1.2], [9, 11.2], [0.9, 1.2])$
Formula:

$$f(x) = \sum_{i=1}^{10} [\exp(-0.1ix_1) - \exp(-0.1ix_2) - (\exp(-0.1i) - \exp(-i))x_3]^2$$

Global minimum: 0
Set of minimizers: $(1, 10, 1)$

5.4 Paviani function

Dimension: $n = 10$
Interval: $X = [2.001, 9.999]^n$

Formula:

$$f(x) = \sum_{i=1}^{n} \left(\ln^2(x_i - 2) + \ln^2(10 - x_i)\right) - \left(\prod_{i=1}^{n} x_i\right)^2$$

Global minimum: -45.778470
Set of minimizers: $(9.350266, 9.350266, ..., 9.350266)$

A function with several local and 1 global minimizer:

5.5 Shekel function

Dimension: $n = 4$
Interval: $X = [0, 10]^n$
Formula:

$$f(x) = -\sum_{i=1}^{m} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

where for $m = 10$:

$A_1 = (4, 4, 4, 4)$	$c_1 = 0.1$	$A_6 = (2, 9, 2, 9)$	$c_6 = 0.6$
$A_2 = (1, 1, 1, 1)$	$c_2 = 0.2$	$A_7 = (5, 5, 3, 3)$	$c_7 = 0.3$
$A_3 = (8, 8, 8, 8)$	$c_3 = 0.2$	$A_8 = (8, 1, 8, 1)$	$c_8 = 0.7$
$A_4 = (6, 6, 6, 6)$	$c_4 = 0.4$	$A_9 = (6, 2, 6, 2)$	$c_9 = 0.5$
$A_5 = (3, 7, 3, 7)$	$c_5 = 0.4$	$A_{10} = (7, 3.6, 7, 3.6)$	$c_{10} = 0.5$

Global minimum: close to -10.0
Set of minimizers: close to $(4, 4, 4, 4)$

5.6 Levy function

Dimension: $n = 4 - 7$
Interval:
for n=4: $X = [-10, 10]^n$
for n=5-7: $X = [-5, 5]^n$
Formula:

$$f(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n))$$

Global minimum:
for n=4: -21.502356
for n=5-7: -11.504403
Set of minimizers:
for n=4: $(1, 1, 1, -9.752356)$
for n=5-7: $(1, ..., 1, -4.754402)$

5.7 Griewank function

Dimension: $n = 10$
Interval: $X = [-500, 700]^n$
Formula:

$$f(x) = \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Global minimum: 0
Set of minimizers: $(0, 0, ..., 0)$

Functions with many local and 9 global minimizers:

5.8 Shubert function

Dimension: $n = 2$
Interval: $X = [-10, 10]^n$
Formula:

$$f(x) = -\sum_{i=1}^{n} \sum_{j=1}^{5} j \sin((j+1)x_i + j)$$

Global minimum: -24.062499
Set of minimizers:

$(-6.774576, -6.774576)$	$(-6.774576, -0.491391)$	$(-6.774576, 5.791794)$
$(-0.491391, -6.774576)$	$(-0.491391, -0.491391)$	$(-0.491391, 5.791794)$
$(5.791794, -6.774576)$	$(5.791794, -0.491391)$	$(5.791794, 5.791794)$

5.9 Hansen function

Dimension: $n = 2$
Interval: $X = [-10, 10]^n$
Formula:

$$f(x) = \sum_{i=1}^{5} i \cos((i-1)x_1 + i) \sum_{j=1}^{5} j \cos((j+1)x_2 + j)$$

Global minimum: -176.541793
Set of minimizers:

$(-7.589893, -7.708314)$	$(-7.589893, -1.425128)$	$(-7.589893, 4.858057)$
$(-1.306708, -7.708314)$	$(-1.306708, -1.425128)$	$(-1.306708, 4.858057)$
$(4.976478, -7.708314)$	$(4.976478, -1.425128)$	$(4.976478, 4.858057)$

References

[1] J.W. Bandler, R. Biernacki, S. Chen, R. Hemmers, and K. Madsen (1995) *Electromagnetic Optimization Exploiting Aggressive Space Mapping.* IEEE Trans. Microwave Theory Tech., MTT-43, 2874-2882.

[2] C.G. Boender and H.E. Romeijn (1995) *Stochastic Methods.* In "Handbook of Global Optimization", Horst and Pardalos, eds. 829-869.

[3] T. Csendes, D. Ratz, (1997) *Subdivision direction selection in interval methods for global optimization* . To appear in in SIAM J. Num. Anal..

[4] E. Hansen (1978) *A global convergent interval analytic method for computing and bounding real roots* BIT 18, 415-424.

[5] E. Hansen (1978) *Global optimization using interval analysis - the multidimensional case* Numerische Mathematik 34, 247-270.

[6] E. Hansen (1992) *Global Optimization using Interval Analysis.* Marcel Dekker, Inc., New York. 230 pages.

[7] H.F. Hansen (1993) *Kombinerede Quasi-Newton og Intervalmetoder til Global Optimering* Report NI-E-93-01, Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark. (In Danish).

[8] K. Madsen (1996) *Real versus Interval Methods for Global Optimization.* Presentation at the Conference "Celebrating the 60th Birthday of M.J.D. Powell", Cambridge, July 1996.

[9] K. Madsen, S. Zertchaninov and Antanas Žilinskas (1998) *Global Optimization using Branch-and-Bound.* Submitted to *Global Optimization.*

[10] R.E. Moore (1966) *Interval Analysis.* Prentice-Hall, 142 pages.

[11] R.E. Moore (1966) *A test for existence of solutions to non-linear systems.* SIAM Journal on Numerical Analysis 14, 611-615.

[12] R.E. Moore (1979) *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 190 pages.

[13] M.J.D. Powell (1970) *A Hybrid Method for Non-Linear Equations.* in "Numerical Methods for Nonlinear Algebraic Equations", P. Rabinowitz, ed. 87-114.

[14] A.G.H. Rinnooy Kan and G.T. Timmer (1989) Global Optimization. In "Handbook in Operations Research Vol 1: Optimization", Nemhauser et.al., eds. 631-659.

[15] S. Skelboe (1974) *Computation of Rational Interval Functions.* BIT 14, 87-95.

[16] S. Zertchaninov and K. Madsen (1998) *A C++ Programme for Global Optimization.* IMM-REP-1998-04, Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.