# Global Optimization by Multilevel Coordinate Search

Waltraud Huyer and Arnold Neumaier
*Institut für Mathematik, Universität Wien*
*Strudlhofgasse 4, A-1090 Wien, Austria*
*(*huyer@cma.univie.ac.at, neum@cma.univie.ac.at*)*

**Abstract.** Inspired by a method by Jones et al. (1993), we present a global optimization algorithm based on multilevel coordinate search. It is guaranteed to converge if the function is continuous in the neighborhood of a global minimizer. By starting a local search from certain good points, an improved convergence result is obtained. We discuss implementation details and give some numerical results.

**Keywords:** Global optimization, bound constraints, local optimization, coordinate search.

**1991 Mathematics Subject Classification:** 90C26.

## 1. Introduction

Problems involving global optimization (traditionally usually minimization) of a multivariate function are widespread in the mathematical modeling of real world systems for a broad range of applications (see, e.g., Pintér (1996)). Many problems can be described only by nonlinear relationships, which introduces the possibility of multiple local minima. The task of global optimization is to find a point where the objective function obtains its smallest value, the global minimum. When the objective function has a huge number of local minima, local optimization techniques are likely to get stuck before the global minimum is reached, and some kind of global search is needed to find the global minimum with some reliability. The global optimization homepage at `http://solon.cma.univie.ac.at/~neum/glopt.html` contains many commented links to online information and software packages relevant to global optimization, and a nice recent online survey of techniques is at `http://www.cs.sandia.gov/opt/survey/`.

Pintér (1996) gives a good overview of methods and software for continuous global optimization. Algorithms for solving global minimization problems can be classified into heuristic methods that find the global minimum only with high probability, and methods that guarantee to find a global optimum with a required accuracy. An important class belonging to the former type are the stochastic methods (e.g., Boender and Romeijn (1995)), which involve function evaluations at a suitably chosen random sample of points and subsequent manipulation of the sample to find good local (and hopefully global) minima. A number of techniques like simulated annealing (e.g., Ingber (1989; 1996)) and genetic algorithms (e.g., Michalewicz (1996)) use analogies to physics and biology to approach the global optimum.

The most important class of methods of the second type are branch and bound methods. They derive their origin from combinatorial optimization (e.g., Nemhauser and Wolsey (1988)), where also global optima are wanted but the variables are discrete and take a few values only. Branch and bound methods guarantee to find a global minimizer with a desired accuracy after a predictable (though often exponential) number of steps. The basic idea is that the configuration space is split recursively by *branch*ing into smaller and smaller parts. This is not done uniformly but instead some parts are preferred and others are eliminated. The details depend on *bound*ing procedures. Lower bounds on the objective allow to eliminate large portions of the configuration space early in the computation so that only a (usually small) part of the branching tree has to be generated and processed. The lower bounds can be obtained by using d.c.-methods (e.g., Horst and Tuy (1996)), techniques of interval analysis (e.g., Hansen (1992)) or majorization resp. minorization methods based on the knowledge of Lipschitz constants (e.g., Pintér (1996)). Unlike heuristic methods, however, these methods are only applicable if something about the analytical properties of the objective is known, since one needs to be able to compute powerful and reliable underestimating functions.

The algorithm we are going to describe in this paper is an intermediate between purely heuristic methods and methods that allow an assessment of the quality of the minimum obtained; it is in spirit similar to the DIRECT method for global optimization by Jones et al. (1993). As the latter method, our method is guaranteed to converge if the objective is continuous in the neighborhood of a global minimizer; no additional smoothness properties are required. In contrast to many stochastic methods that operate only at the global level and are therefore quite slow, our algorithm contains local enhancements that lead to quick convergence once the global part of the algorithm has found a point in the basin of attraction of a global minimizer. Moreover, for all control variables in our algorithm meaningful default values can be chosen that work simultaneously for most problems.

In this paper, we consider the bound constrained optimization problem

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } \ x \in [u, v] \end{aligned} \tag{1}$$

with finite or infinite bounds, where we use interval notation for rectangular boxes,

$$[u,v] := \{x \in I\!R^n \mid u_i \le x_i \le v_i,\ i = 1,\dots,n\},$$

with $u$ and $v$ being $n$-dimensional vectors with components in $\overline{I\!R} := I\!R \cup \{-\infty,\infty\}$ and $u_i < v_i$ for $i = 1,\dots,n$, i.e., only points with finite components are regarded as elements of a box $[u,v]$ whereas its bounds can be infinite. In the case where all bounds are infinite we obtain an unconstrained optimization problem.

In DIRECT, a finite box is normalized to $[0,1]^n$ and partitioned into smaller boxes. Each box is characterized by its midpoint, and the side lengths of the boxes are always of the form $3^{-k}$, $k \in I\!N_0$. A disadvantage of that algorithm is that infinite box bounds cannot be handled. Moreover, since the boundary can never be reached, it converges more slowly than necessary in cases where the minimizer lies at the boundary. For example, in the case of functions that are monotonous in each variable the optimizer is at a vertex, but DIRECT converges unnecessarily slowly. Inspired by DIRECT, we devised a global optimization algorithm, where we remedy the above shortcomings and allow for a more irregular splitting procedure. Our algorithm is based on *m*ultilevel *c*oordinate *s*earch and we therefore call it MCS. Note that this has nothing to do with multilevel optimization (cf. Vicente and Calamai (1994) and Migdalas et al. (1998), for example); we use the word 'multilevel' in a different sense. Moreover, the multilevel search algorithm by Goertzel (1992) also uses a different notion of levels.

In Section 2, an outline of our implementation of the MCS algorithm is given, and details are explained in Sections 3 to 5. In Section 6 we prove the convergence of our algorithm. Finally, numerical results are presented in Section 7. Throughout the text, inequalities, absolute values, max and min for vectors are interpreted in their natural componentwise meaning.

## 2. The MCS Algorithm

We first give an overview of the ideas behind MCS and leave the discussion of the details to Sections 3 to 5. There are many ways to design algorithms based on these ideas and guaranteeing convergence to a global minimizer (cf. Section 6). But trivial implementations are very slow, and a number of heuristic enhancements are needed to obtain a high quality method. We tried several variants; in the following we describe a specific version that performed well in our tests.

As in DIRECT, we try to find the minimizer by splitting the search space into smaller boxes. Each box contains a distinguished point, the so-called *base point*, whose function value is known. The partitioning procedure is not uniform but parts where low function values are expected to be found are preferred. Since interval subdivision is also part of what is done in branch and bound methods, our algorithm can be regarded as *branch without bound.*

Like DIRECT, our algorithm combines global search (splitting boxes with large unexplored territory) and local search (splitting boxes with good function values). The key to balancing global and local search is the multilevel approach. As a rough measure of the number of times a box has been processed, a *level* $s \in \{0, 1, \ldots, s_{\max}\}$ is assigned to each box. Boxes with level $s_{\max}$ are considered too small for further splitting; a level $s = 0$ indicates that a box has already been split and can be ignored. Whenever a box of level $s$ $(0 < s < s_{\max})$ is split, its level is set to zero, and its descendants get level $s+1$ or $\min(s+2, s_{\max})$. Thus the levels of MCS correspond to the side lengths of DIRECT, i.e., the boxes with small level are the 'large' boxes that have not been split very often yet. After an initialization procedure, the algorithm proceeds by a series of *sweeps* through the levels (cf. Section 3). The fact that we start with the boxes at the lowest levels in each sweep constitutes the global part of the algorithm, and at each level the box with lowest function value is selected, which forms the local part of the algorithm.

Ratz and Csendes (1995) and Csendes and Ratz (1997) investigated a number of rules for selecting an optimal component to bisect a box in branch and bound methods and called these rules interval subdivision direction selection rules. The simplest rule is the interval-width-oriented rule, which divides the original box in a uniform way and was also used by Jones et al. (1993). However, heuristic direction selection rules dependent on information about the variability of $f$ in the different coordinates may yield an improvement; cf. also Hansen (1992). In contrast to DIRECT, which usually splits a box along several coordinates, we split along a single coordinate in each step. Information gained from already sampled points is used to determine the splitting coordinate as well as the position of the split. Usually a single new function evaluation is needed to split a box into two or even three subboxes. The base points of the descendants of a box are chosen such that they differ from the base point of the parent box in (at most) one coordinate. Thus this procedure of generating new function values is a variant of the standard coordinate search method. Safeguards are incorporated to prevent splits that are too asymmetric. They ensure that the descendants of a bounded box will eventually become arbitrarily small after sufficiently many splits

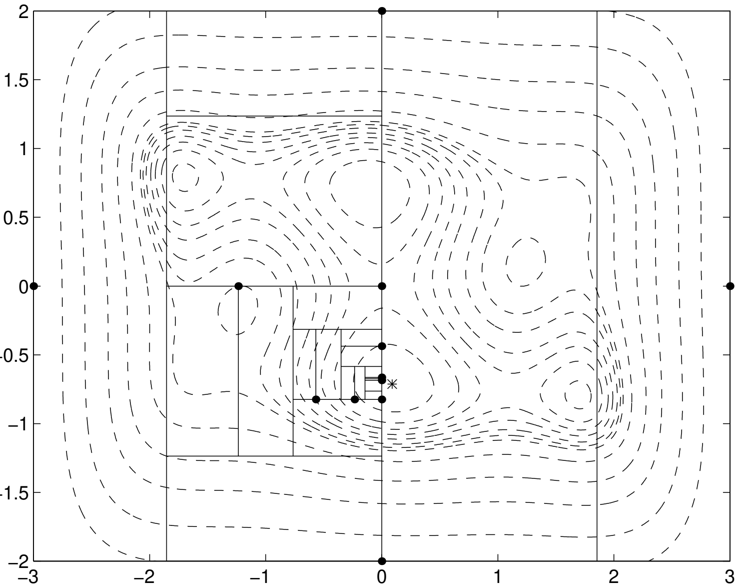


*Figure 1.* Result of MCS for the six-hump camel function.

along each coordinate and that the descendants of an unbounded box will also shrink sufficiently fast.

*MCS without local search* puts the base points and function values of boxes of level $s_{\max}$ into the so-called *shopping basket* (containing 'useful' points). *MCS with local search* tries to accelerate convergence of the algorithm by starting local searches from these points before putting them into the shopping basket. More precisely, we first check whether the base point of a newly generated box of level $s_{\max}$ is likely to be in the basin of attraction of a local minimizer already in the shopping basket; only if this is not the case, we start a local search from it. The local search algorithm used in our implementation of MCS essentially consists of building a local quadratic model by *triple searches*, then defining a promising search direction by minimizing the quadratic model on a suitable box and finally making a line search along this direction.

In Figure 1 we show the boxes obtained by MCS for the six-hump camel function with default box bounds (cf. Section 7). The base points are indicated by fat dots; the asterisk denotes the point obtained by local optimization and turns out to be a global optimizer. The dashed lines are the contour lines of the function, where the six humps are clearly discernible.

## 3. Initialization and Sweeps

Unlike in DIRECT, a base point can belong to more than one box, and the base point of a box in our algorithm is usually not the midpoint but a point at the boundary, often but not always a vertex. Moreover, we also assign to each box an *opposite point.* The construction is such that the base point $x$ and the opposite point $y$ determine the box, and we call such a box $B[x, y]$.

The algorithm starts with a so-called *initialization procedure* producing an initial set of boxes. Whenever a box is split along some coordinate $i$ for the first time (either in the initialization procedure or later), this is done at three or more user-defined values $x_i^l$ (where function values are computed) and some adaptively chosen intermediate points, and at least four subboxes are obtained. More precisely, let

$$u_i \le x_i^1 < x_i^2 < \ldots < x_i^{L_i} \le v_i, \quad L_i \ge 3, \quad i = 1, \ldots, n,$$

be given. We first evaluate $f$ at an initial point $x^0$ and set $x^* = x^0$. Then, for $i = 1, \ldots, n$, $f$ is evaluated at $L_i - 1$ points in $[u, v]$ that agree with $x^*$ in all coordinates $k \ne i$. Thus we have $L_i$ pairs $(x^l, f_i^l)$ $(l = 1, \ldots, L_i)$ with

$$\begin{aligned} x_k^l &= x_k^* \ (k \ne i), \\ f_i^l &= f(x^l), \end{aligned}$$

and $x^* = x^{l_i}$ for some $l_i$. The point with smallest function value is then renamed $x^*$ before repeating the procedure with the next coordinate. The numbers $x_i^l$, $l = 1, \ldots, L_i$, and the indices $l_i$ are stored in an *initialization list.* The choice of the initialization list is left to the user, who may incorporate in it knowledge about the likely distribution of good points. A good guess for the global optimizer can be used as $x^0$. Some possible choices are discussed in Section 7.

From the initialization list and the corresponding list of function values, an initial set of boxes is constructed as follows. The root box is $B[x, y] = [u, v]$, with $x = x^0$ as base point and as $y$ one of the corners of $[u, v]$ farthest away from $x$. Note that $x$ need not be a vertex and that some or all coordinates of $y$ can be infinite. For $i = 1, \ldots, n$, the current box is split along the $i$th coordinate into $2L_i - 2$, $2L_i - 1$ or $2L_i$ subintervals with exactly one of the $x_i^l$ as endpoints, depending on whether two, one or none of the $x_i^l$ are on the boundary, which means that in addition to $x_i^l$, $l = 1, \ldots, L_i$, we have to split at $z_i^l$, $l = 2, \ldots, L_i$. The additional splitting points are chosen as $z_i^l = x_i^{l-1} + q^k(x_i^l - x_i^{l-1})$, $l = 2, \ldots, L_i$, where $q = \frac{1}{2}(\sqrt{5} - 1)$ is the golden section ratio and $k = 1$ or 2 is chosen such that the part with the smaller function value gets the larger fraction of the interval. The resulting subboxes get as base

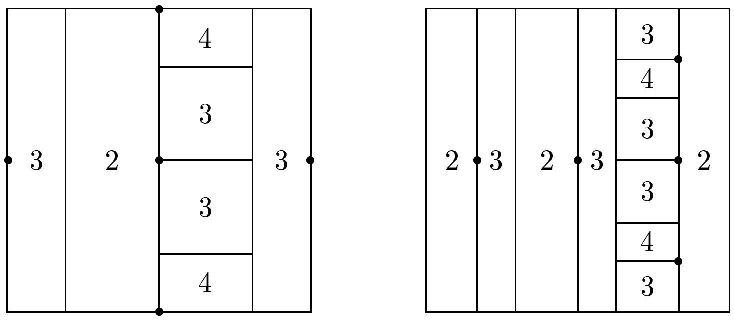


*Figure 2.* Examples for the initialization procedure.

point the point $x'$ obtained from the current $x = x^*$ by changing $x_i$ to the $x_i^l$ that is a boundary point of the corresponding $i$th coordinate interval, so that $f(x') = f_i^l$, and as opposite point the point obtained from $y$ by changing $y_i$ to the other end of that interval.

The information available so far allows us to define priorities on the coordinates. For each $i$, we compute the union of the ranges of the quadratic interpolant through any three consecutive $(x_i^l, f_i^l)$ and take the difference of the upper and lower bound obtained as a crude measure of the variability of $f$ with the $i$th component. Components with higher variability get a higher priority, and this ranking is saved in a vector $\pi$ such that the component with index $i$ has the $\pi_i$th highest estimated variability. Moreover, if the $x^*$ obtained after splitting the $i$th coordinate belongs to two boxes, the one containing the minimizer of the quadratic models is taken as current box for coordinate $i+1$.

The root box gets level 1. When a box of level $s$ is split, the boxes with the smaller fraction of the golden section split get level $s+2$ and all other boxes get level $s+1$. Thus the current box for splitting in the next coordinate is in any case one with level $s+1$, and after finishing the initialization procedure, the first level is empty and the non-split boxes have levels $2, \dots, n+2$, which implies that it is meaningful to take $s_{\max} \gg n$. Two examples for the set of boxes, their base points and their levels after the initialization procedure in the two-dimensional case are shown in Figure 2, where $x_i^1 = u_i$, $x_i^2 = \frac{1}{2}(u_i + v_i)$, $x_i^3 = v_i$ (left-hand side) and $x_i^1 = \frac{5}{6}u_i + \frac{1}{6}v_i$, $x_i^2 = \frac{1}{2}(u_i + v_i)$, $x_i^3 = \frac{1}{6}u_i + \frac{5}{6}v_i$ (right-hand side); in both cases we have $x^0 = \frac{1}{2}(u + v)$.

It is easy to see the connection between the golden section split and the assignment of levels. When a box $B$ with level $s$ is split along coordinate $i$ according to the golden section split and its larger part $B'$ is again split according to the golden section split along the $i$th coordinate, the larger descendant of $B'$ has the same $i$th coordinate length as the smaller descendant of $B$, and these boxes both have level $s + 2$. Moreover, the box with the better function value gets the larger fraction of the interval and the smaller level because then it is more likely to be split again more quickly, which was also the strategy adopted in DIRECT.

Any choice of $x_i^l$ including the endpoints $u_i$, $v_i$ in the list guarantees that, in the simple case that $f$ is monotonous in each variable, the final $x^*$ of the initialization phase is already the global minimizer.

At least three values of $x_i^l$ are needed in order to determine the ranks $\pi_i$ by quadratic interpolation. It is reasonable to make the first splits along each coordinate at some predetermined values since heuristic determination of an optimal split of a wide interval would not be very reliable anyway.

After the initialization procedure, the branching process proceeds by a series of *sweeps* through the levels. A sweep is defined by the following three steps.

STEP 1. Scan the list of non-split boxes and define a *record list* containing for each level $0 < s < s_{\max}$ a label $b_s$ pointing to a box with the lowest function value among all boxes at level $s$. If there is no box at level $s$, set $b_s = 0$. Initialize $s$ to the lowest level with $b_s \neq 0$.

STEP 2. The box with label $b_s$ is a candidate for splitting. If the box is not split (according to the rule given in Section 4), its level is increased by one and possibly $b_{s+1}$ has to be updated. If the box is split, mark it as split and insert its children. Update the record list if any of the children yields a strict improvement of $f$ on its level.

STEP 3. Increase $s$ by 1. If $s = s_{\max}$, start new sweep. Else if $b_s = 0$ go to Step 3, else go to Step 2.

Clearly, each sweeps ends after at most $s_{\max} - 1$ visits of Step 3.

## 4. Splitting

Instead of storing the box bounds for each box we store information on its history (label of the parent box, splitting index, splitting value, a label identifying which of the many children it is etc.). This keeps the amount of storage proportional to the number of function evaluations and allows us to recover information to build a separable quadratic model by going back in the history of the box. When a box of level $s < s_{\max}$ is a candidate for splitting (cf. Section 3), we recover its base point $x$, the opposite point $y$ and the number $n_j$ of times coordinate $j$ has been split in the history of the box. We distinguish two cases.

CASE 1 (*splitting by rank*). If

$$s > 2n(\min n_j + 1), \qquad (2)$$

the box is always split, and the splitting index is a coordinate $i$ with $n_i = \min n_j$.

CASE 2 (*splitting by expected gain*). Otherwise, the box may be split along a coordinate where the maximal gain in function value is expected according to a local separable quadratic model obtained by fitting $2n+1$ function values. However, if the expected gain is not large enough, the box is not split at all but its level is increased by one.

A box that is not eligible for splitting by expected gain will eventually reach level $2n(\min n_j + 1) + 1$ and be split by rank provided $s_{\max}$ is large enough. Thus, for $s_{\max} \to \infty$, the splitting by rank rule guarantees that each coordinate is split arbitrarily often. (Many other thresholds in place of (2) would provide the same conclusion; cf. also Section 6.)

In order to handle correctly the adaptive splitting points and unbounded intervals we introduce some more notation. Suppose that we want to split the $i$th coordinate interval

$$\Box\{x_i, y_i\} := [\min(x_i, y_i), \max(x_i, y_i)]$$

for $x_i \in \mathbb{R}$, $y_i \in \overline{\mathbb{R}}$, and suppose that $x_i$ is the $i$th component of the base point of the box being considered. Since the descendants of a box should shrink sufficiently fast, we may not split too close to $x_i$. If $y_i$ is large, we also do not want the new component $x_i'$ to be too large and therefore force it to belong to some smaller interval $\Box\{\xi', \xi''\}$. We choose this interval according to

$$\xi'' := \mathrm{subint}(x_i, y_i), \quad \xi' := x_i + (\xi'' - x_i)/10, \tag{3}$$

where

$$\mathrm{subint}(x, y) := \begin{cases} \mathrm{sign}(y) & \text{if } 1000|x| < 1,\ |y| > 1000, \\ 10\,\mathrm{sign}(y)|x| & \text{if } 1000|x| < 1,\ |y| > 1000|x|, \\ y & \text{otherwise.} \end{cases} \tag{4}$$

We are now ready to describe the two splitting rules in more detail.

### 4.1. Splitting by Rank

Let $s > 2n(\min n_j + 1)$. This means that, although the box has already reached a rather high level, there is at least one coordinate along which the box has not yet been split very often. Then we select the splitting index $i$ among the indices $i$ with smallest $n_i$ as the one with lowest $\pi_i$ (and hence highest variability rank). The name 'splitting by rank' thus refers to the ranking of the coordinates by $n_i$ and $\pi_i$.

If $n_i = 0$, the splitting is done according to the initialization list at $x_i^l$, $l = 1, \ldots, L_i$, and at the golden section split points, as discussed in Section 3, and the new base points and opposite points are defined as before. The boxes with the smaller fraction of the golden section split (and thus larger function values) get level $\min(s+2, s_{\max})$, and all other ones get level $s+1$.

If $n_i > 0$, the $i$th component ranges between $x_i$ and $y_i$, and the splitting value is chosen as $z_i = x_i + \frac{2}{3}(\xi'' - x_i)$, where $\xi'' := \mathrm{subint}(x_i, y_i)$ is given by (4). The box is split at $z_i$ and at the golden section split point, and we obtain three parts with only one additional function evaluation at the point $x'$ obtained by changing the $i$th coordinate of $x$ to $z_i$. The smaller fraction of the golden section split gets level $\min(s+2, s_{\max})$, and the two other parts get level $s+1$. Moreover, the base point of the first child is taken to be $x$, the base point of the second and third child is the point $x'$ defined above, and the opposite points are obtained by changing $y_i$ to the other end of the $i$th coordinate interval of the corresponding box. Since this split is mainly made to reduce the size of a large interval and not for an expected large reduction in function value, we do not try to determine an optimal $z_i$ but take a value that is predetermined by the box bounds. The factor $\frac{2}{3}$ is motivated by the fact that the box is split into three parts, where the second split is made between $x_i$ and $z_i$.

### 4.2. Splitting by Expected Gain

If $s \leq 2n(\min n_j + 1)$, we determine the optimal splitting index and position of the split from a local separable quadratic model, which is a reasonably simple local approximation of $f$. To this end we need two additional points and corresponding function values for each coordinate. Whenever we have split in the $i$th coordinate in the history of the box, we obtain values that can be used for quadratic interpolation in this coordinate. For each coordinate we take the first two points and function values found by pursuing the history of the box back to $[u, v]$ since these points are expected to be closest to the base point $x$. For coordinates that have not yet been split, we obtain this information from the initialization list. Let

$$e(\xi) = f(x) + \sum_{i=1}^{n} e_i(\xi_i)$$

be the local separable model for $f(\xi)$ generated by interpolation at $x$ and the $2n$ additional points collected as above. For each coordinate $i$, we define the *expected gain* $\hat{e}_i$ in function value when we evaluate at a new point obtained by changing this coordinate in the base point. Again two cases have to be distinguished.

CASE 1. In the history of the current box, coordinate $i$ was never split, i.e., $n_i = 0$. Then we split according to the initialization list at points where we already know the obtainable function differences, and therefore compute the expected gain as

$$\hat{e}_i = \min\{f_i^l \mid l = 1, \ldots, L_i\} - f_i^{l_i}.$$

CASE 2. If $n_i > 0$, the $i$th component ranges between $x_i$ and $y_i$, and with the quadratic partial correction function

$$e_i(\xi_i) = \alpha_i(\xi_i - x_i) + \beta_i(\xi_i - x_i)^2$$

at our disposal, we can calculate an approximation to the maximal gain expected when changing the value of $x_i$ only. For the reasons discussed above, we choose the splitting value from the interval $\square\{\xi', \xi''\}$ defined by (3). Then we compute

$$\hat{e}_i = \min_{\xi_i \in \square\{\xi', \xi''\}} e_i(\xi_i)$$

with minimum achieved at $\xi_i = z_i$. If the expected best function value satisfies

$$f_{\text{exp}} := f(x) + \min_{1 \leq i \leq n} \hat{e}_i < f_{\text{best}}, \tag{5}$$

where $f_{\text{best}}$ is the current best function value (including the function values obtained by local optimization), we expect the box to contain a better point and split, using as splitting index the component with minimal $\hat{e}_i$. Condition (5) prevents wasting function evaluations by splitting boxes with bad base point function values; these boxes will eventually be split by rank anyway.

In Case 1 we again split according to the initialization list, and the definition of the new base points and opposite points and the assignment of levels are as before. In Case 2 we use $z_i$ as splitting value and the box is split at $z_i$ (if $z_i \neq y_i$) and at the golden section split point, and we obtain two or three parts. The larger fraction of the golden section split gets level $s + 1$, the smaller fraction level $\min(s + 2, s_{\max})$. If $z_i \neq y_i$ and the third part is larger than the smaller fraction of the golden section split, it gets level $s + 1$; otherwise it gets level $\min(s + 2, s_{\max})$. Moreover, the base point of the first child is taken to be $x$, the base point of the second and third (if $z_i \neq y_i$) child is obtained by changing the $i$th coordinate of $x$ to $z_i$, and the opposite points are again obtained by changing $y_i$ to the other end of the $i$th coordinate interval of the box.

If (5) is violated, we do not expect any improvement and therefore do not split but increase the level by 1.

## 5. Local Search

The theory of local optimization provides powerful tools for the task of optimizing a smooth function when knowledge of the gradient or even the Hessian is assumed. When no derivative information is available but the function is known to be twice continuously differentiable, the traditional methods are based on the employment of successive line searches, using search directions defined by minimizing quadratic models built from conjugate directions; cf. the direction set method of Powell (1964) and a modification due to Brent (1973). These algorithms, however, do not allow the specification of bound constraints. Elster and Neumaier (1995) developed an algorithm for optimization of low-dimensional bound constrained functions, based on the use of quadratic models and a restriction of the evaluation points to successively refined grids. However, the work in that algorithm grows with the dimension $n$ as $O(n^6)$ and hence is unsuitable for larger dimensions.

The local optimization algorithm we are going to describe in the sequel also makes use of quadratic models and successive line searches, and devices to handle bound constraints are incorporated. We first explain two important ingredients of our local search algorithm: the procedure of building a local quadratic model by triple searches and the coordinate search procedure.

### 5.1. Triple Search

While quadratic models are most appropriate when the function to be optimized is smooth enough, they seem to be useful also in general (cf. the two discontinuous examples in Table VI below). We want to use $\binom{n+2}{2}$ function values to construct a quadratic model

$$q(x) = f + g^T(x - x^{\text{best}}) + \frac{1}{2}(x - x^{\text{best}})^T G(x - x^{\text{best}}),$$

which is hoped to be a good local approximation of $f$. The triple search does not only aim at constructing a quadratic model but also at reducing the function value.

Assume that we have three vectors $x^l < x^m < x^r$. The function values are to be taken at points $x$ with $x_i \in \{x_i^l, x_i^m, x_i^r\}$ (hence the name 'triple search') as follows. If $x^{\text{best}}$ denotes current best point in the triple search, denote by $x^{(i,1)}$ and $x^{(i,2)}$ the points obtained from $x^{\text{best}}$ by changing its $i$th coordinate to the other two values in $\{x_i^l, x_i^m, x_i^r\}$, and by $x^{ik}$, $k < i$, the points obtained by changing the $i$th and $k$th coordinate to the ones with the smaller $q(x^{(i)})$ resp. $q(x^{(k)})$ with the current quadratic model $q$. Thus we obtain the following procedure that we are going to describe in more detail in the sequel:

$f = f(x^{\text{best}})$
**for** $i = 1$ **to** $n$
  compute $f(x^{(i,1)})$ and $f(x^{(i,2)})$; compute $g_i$ and $G_{ii}$
  store $x^{\text{newbest}}$ but do not update $x^{\text{best}}$
  **for** $k = 1$ **to** $i-1$
    compute $q(x^{(k,1)})$ and $q(x^{(k,2)})$ from the current model
    compute $f(x^{ik})$
    update $x^{\text{newbest}}$ but do not update $x^{\text{best}}$
    compute $G_{ik}$
  **end for**
  **if** $x^{\text{newbest}} \neq x^{\text{best}}$, update $x^{\text{best}}$, $f$ and $g_{1:i}$; **end if**
**end for**

When, for an $i$, $1 \leq i \leq n$, we have already computed approximations for $g_l$ and $G_{lk}$, $l = 1, \ldots, i-1$, $k = 1, \ldots, l$, by interpolating at $\binom{i+1}{2}$ points that differ only in the first $i-1$ components, and $x^{\text{best}}$ is the current best point in the triple search, we obtain approximations for $g_i$ and $G_{ii}$ by determining these numbers such that the quadratic polynomial

$$p(\xi) = f(x^{\text{best}}) + g_i(\xi - x_i^{\text{best}}) + \frac{1}{2}G_{ii}(\xi - x_i^{\text{best}})^2$$

interpolates at $(x_i^{(i,j)}, f(x^{(i,j)}))$, $j = 1, 2$. If $\min(f(x^{(i,1)}), f(x^{(i,2)})) < f(x^{\text{best}})$, we may not yet update $x^{\text{best}}$ (this would invalidate the previous computations), but we store the new best point as $x^{\text{bestnew}}$.

Assume that, in addition, we have already calculated approximations for $G_{il}$, $1 \leq l \leq k-1$, for a $k$, $1 \leq k \leq i-1$. Then we can compute

$$q(x^{(k,j)}) = f(x^{\text{best}}) + g_k(x_k^{(k,j)} - x_k^{\text{best}}) + \frac{1}{2}G_{kk}(x_k^{(k,j)} - x_k^{\text{best}})^2, \quad j = 1, 2,$$

with the current quadratic model $q$, and we have $q(x^{(i,j)}) = f(x^{(i,j)})$, $j = 1, 2$. Let $x^{ik}$ be defined as above. Then we choose $G_{ik} = G_{ki}$ such that the quadratic model interpolates at $x^{ik}$, i.e., such that the equation

$$\begin{aligned} f(x^{ik}) \;=\; & f(x^{\text{best}}) + g_k(x_k^{ik} - x_k^{\text{best}}) + g_i(x_i^{ik} - x_i^{\text{best}}) \\ & + \frac{1}{2}G_{kk}(x_k^{ik} - x_k^{\text{best}})^2 + G_{ik}(x_k^{ik} - x_k^{\text{best}})(x_i^{ik} - x_i^{\text{best}}) \\ & + \frac{1}{2}G_{ii}(x_i^{ik} - x_i^{\text{best}})^2 \end{aligned}$$

is satisfied. Again we do not update $x^{\text{best}}$ but update $x^{\text{bestnew}}$ if $x^{ik}$ yields a strict improvement in function value.

After finishing the loop over $k$, we reexpand the model around the new best point by

$$g_k = g_k + \sum_{l=1}^{i} G_{kl}(x_l^{\text{bestnew}} - x_l^{\text{best}}), \quad k = 1, \ldots, i,$$
$$x^{\text{best}} = x^{\text{bestnew}}, \quad f = f(x^{\text{bestnew}}).$$

It is easy to see that the above method gives the unique quadratic interpolant to $f$ at $\binom{n+2}{2}$ distinct points; in particular, we recover the exact objective function as $q(x) = f(x)$ whenever $f$ is quadratic.

An optional input parameter handles the case where *knowledge about the sparsity pattern* of the Hessian is assumed. If we know that $G_{ik}$ is zero for some $k < i$, we can omit a step in the inner for-loop of the triple search procedure. For sufficiently sparse Hessians, this saves a large fraction of function values spent in the local optimizations.

In a *diagonal triple search* we only carry out the diagonal part of the above algorithm and take the off-diagonal elements of the Hessian from the previous iteration; thus only $2n$ additional function values are needed.

### 5.2. Coordinate Search

In order to find $x^l, x^m, x^r$ for the triple search, we sometimes use a *coordinate search* based on a line search routine. A MATLAB 4.2 version of the actual line search used can be obtained electronically from `http://solon.cma.univie.ac.at/~neum/software/gls`. The univariate line search program `gls` contains a parameter `smaxls` limiting the number of points used for the line search. It is possible to feed other points in addition to the starting point and their function values into the program, and these points are included in `smaxls`.

A line search with `smaxls` $= 6$ is made along each coordinate. The first line search is started with the candidate for the shopping basket. After the line search along the first coordinate, we can take the best point and its two nearest neighbors on both sides (or, if such points do not exist, the two nearest neighbors on one side) as $\{x_1^l, x_1^m, x_1^r\}$. The subsequent line searches are started with the current best point obtained from the previous line search. After a line search in a coordinate $i > 1$, we take the best point, the starting point of the line search (if it is different from the best point) and, if possible, the nearest neighbor of the best point on the other side as $\{x_i^l, x_i^m, x_i^r\}$. The $i$th coordinate of the old $x^{\text{best}}$ has to be among $\{x_i^l, x_i^m, x_i^r\}$ since otherwise we would lose all points through which the surface has been fitted previously.

### 5.3. Local Search

Now we have all ingredients at our disposal to describe the steps of the local search algorithm used in our implementation of MCS.

In Step 1 below we start with looking for better points without being too local yet and therefore determine $x^l, x^m, x^u$ for the triple search by a coordinate search. In Step 2 we calculate the minimizer of the quadratic model, hoping that it yields a good point, and make a line search along this direction. Step 3 controls the loop over the subsequent iterations of building a quadratic model by a triple search (Step 5) and making a line search (Step 6). The difference between Step 1 and Step 5 is that in Step 5 the values for the triple search are taken at small distances $\delta$. Step 4 takes care of the boundary.

In the following $x$ and $f$ always denote the current point and its function value, respectively.

Step 1. Starting with the candidate for the shopping basket, we make a full triple search, where $x^l, x^m, x^r$ are found by a coordinate search as described above. This procedure yields a new point $x$, its function value $f$, an approximation $g$ of the gradient and an approximation of the Hessian $G$.

Step 2. Initially we take $d := \min(v - x, x - u, 0.25(1 + |x - x_0|))$, where $x_0$ is the absolutely smallest point in $[u, v]$, and minimize the quadratic function $q(h) := f + g^T h + h^T G h$ over the box $[-d, d]$. Then we make a line search with `gls` along $x + \alpha p$, where $p$ is the solution of the minimization problem. The values $\alpha = 0$ and $\alpha = 1$ are used as input for `gls`, and the new point and function value are again denoted by $x$ and $f$, respectively. Set $r := (f_{\text{old}} - f)/(f_{\text{old}} - f_{\text{pred}})$, where $f_{\text{old}}$ is the function value of the current point at the beginning of Step 2 and $f_{\text{pred}}$ is the function value predicted by the above quadratic model with $\alpha = 1$. Since $r = 1$ if $f = f_{\text{pred}}$, the deviation of $r$ from 1 measures the predictive quality of the quadratic model.

Step 3. Stop if some limit on the number of visits to Step 3 (per local search) or the limit on function calls has been exceeded. Also stop if none of the components of the current $x$ are at the boundary, the last triple search was a full one and a stopping criterion (see below) is fulfilled.

Step 4. If some components of the current $x$ are at the boundary and the stopping criterion is fulfilled, we make line searches with at most `smaxls` points along these coordinates. If the function value was not improved by these coordinate searches, stop.

Step 5. If $|r - 1| > 0.25$ or the stopping criterion was fulfilled in Step 3, we make a full triple search, otherwise we make a diagonal triple search. We make these triple searches only in the coordinates $i$ such that the component $x_i$ of the current $x$ is not at the boundary. The set $\{x_i^l, x_i^m, x_i^r\}$ is taken to consist of $x_i$, $\max(x_i - \delta, u_i)$ and $\min(x_i + \delta, v_i)$ (resp. two neighbors at distance $\delta$ and $2\delta$ if $x_i$ lies at the boundary) and $\delta = \sqrt[3]{\varepsilon}$, where $\varepsilon$ denotes the machine accuracy. We obtain a new point $x$ and an approximation of its reduced gradient $g$ and its reduced Hessian $G$.

STEP 6. If $r < 0.25$, the last quadratic model was not very good and we therefore shrink the box for minimization by setting $d = d/2$. On the other hand, if $r > 0.75$, the search direction from the quadratic model was rather good and we set $d = 2d$. We now minimize the quadratic function $q$ given in Step 2 over $[\max(-d, u-x), \min(d, v-x)]$ and make a line search along $x + \alpha p$, where $p$ is the solution of the minimization problem, with $\alpha = 0$ and $\alpha = 1$ as input as before. The quantity $r$ is defined as in Step 2, where now $f_{\text{old}}$ is the function value at the current point at the beginning of Step 6, and go to Step 3.

The stopping criterion in Step 3 is defined to be fulfilled if the function value has not been improved by Steps 5 and 6 (resp. Steps 1 and 2) or if the approximated gradient is small, in the sense that $|g|^T \max(|x|, |x_{\text{old}}|) < \gamma(f - f_0)$, where $\gamma$ is an input parameter of our program and $f_0$ is the smallest function value found in the initialization procedure. When some components of the current $x$ are at the boundary and the stopping criterion is fulfilled, we make an attempt in Step 4 to get away from the boundary, and when we are not successful, the algorithm stops.

If the ratio $r$ of the actual to the predicted gain in function value in the previous step is far away from 1, this is an indicator that the quadratic model was not very good and we therefore make a full triple search. However, when $r$ is reasonably close to 1, the last quadratic model was a rather good approximation and we therefore save function values by taking over the off-diagonal elements of the Hessian from the previous quadratic model. In Step 2, the quadratic model is minimized over a not too large initial box, and the box is made larger or smaller depending on $r$.

In Steps 2 and 6, we use a routine for indefinite quadratic programming with bound constraints since there is no guarantee that $G$ is positive definite. A MATLAB 4.2 program `minq` for doing this is available from `http://solon.cma.univie.ac.at/~neum/software/minq/`.

### 5.4. SHOPPING BASKET

The local searches are only carried out at the end of each sweep. Then all candidates for the shopping basket (i.e., all base points of boxes with level $s_{\max}$) that have been collected in this sweep are sorted by ascending function value. For each candidate $x$ for the shopping basket, we check the monotonicity properties of $f$ between $x$ and any point $w$ already in the shopping basket by evaluating $f$ at two uniformly spaced points between $x$ and $w$ to avoid unnecessary local optimization. The procedure is described by the following four steps, where the letter $x$ is used for candidates for the shopping basket (and their updates) and $w$ denotes points already in the shopping basket. In Step 4, a new $w$ may be added, which has to be considered in the subsequent iterations of Steps 1 to 4.

STEP 1. We first check whether we have already made a local search from $x$. (Often a point belongs to two boxes.) If this is the case, we take the next $x$ and go to Step 1.

Otherwise, let the points $w$ already in the shopping basket be sorted by their distance to $x$, starting with the nearest point. For each $w$ such that $f(w) \leq f(x)$ we do the following.

STEP 2. Compute the function value at $x' = x + \frac{1}{3}(w - x)$. If $f(x') > f(x)$, $x$ does not lie in the domain of attraction of $w$. Take the next point $w$ and go to Step 2.

STEP 3. Compute the function value at $x'' = x + \frac{2}{3}(w - x)$. If $f(x'') > \max(f(x'), f(w))$, set $x = x'$ if $f(x') < f(x)$, take the next point $w$ and go to Step 2. Else if $\min(f(x'), f(x'')) < f(w)$, all four points seem to lie in the same valley. However, we do not discard $x$ for local search but set $x$ to the value $x'$ or $x''$ with the smaller function value, take the next point $w$ and go to Step 2. Else (there seems little point to start a local search from $x$ because the four function values are monotonous) we take the next $x$ and go to Step 1.

STEP 4. If a point $x$ (resp. its update) survives the loop over Steps 2 and 3, a local search is started. Subsequently, we apply a procedure similar to Steps 2 and 3 to the result $x$ of local search in order to find out whether we have really found a new point, and only in this case $x$ is put into the shopping basket.

## 6. Convergence of the Algorithm

If the number of levels $s_{\max}$ goes to infinity, MCS is guaranteed to converge to the globally optimal function value if the objective function is continuous – or at least continuous in the neighborhood of a global optimizer. This follows from the fact that then the set of points sampled by MCS forms a dense subset of the search space. That is, given any point $x \in [u, v]$ and any $\delta > 0$, MCS will eventually sample a point within a distance $\delta$ from $x$.

Indeed, in each sweep one box leaves the lowest non-empty level and no box is added at that level. Each level will eventually become empty; in particular the splitting procedure will come to an end when all non-split boxes have level $s_{\max}$. Condition (2) for splitting by rank together with the fact that by splitting the levels advance by at most two guarantees that each box with level $s > 2n(m+1)+2mn$, $m \in \mathbb{N}_0$, has been split at least $m$ times in each coordinate (proof by induction over $m$). Moreover, the safeguards against too narrow splits guarantee that the boxes containing any point $x \in [u,v]$ shrink sufficiently fast after sufficiently many splits. More precisely, for each $\delta > 0$ and $i = 1,\ldots,n$, there exists an $m_i(\delta) \in \mathbb{N}$ such that the $i$th side length of the box containing $x$ is less than $\delta$ if it has been split at least $m_i(\delta)$ times along the $i$th coordinate.

These properties give the following convergence theorem for MCS without local search.

THEOREM 1. *Suppose that the global minimization problem (1) has a solution $\hat{x} \in [u,v]$, and that $f : [u,v] \to \mathbb{R}$ is continuous in a neighborhood of $\hat{x}$, and let $\varepsilon > 0$. Then there exists an $s_0$ such that for each $s_{\max} \geq s_0$, the algorithm will eventually find a base point $x$ with $f(x) < f(\hat{x}) + \varepsilon$; i.e., the algorithm converges if the number of levels tends to $\infty$.*

The worst case of ‘eventually’ in the above theorem is reached when all levels $s < s_{\max}$ are empty and the algorithm is done. It is not difficult to show that the number of sweeps needed for finding such a point is at most $(p^{s_0-1} - 1)/(p-1)$, where $p$ is the upper bound on the number of boxes generated in one splitting step (determined by the initialization list since a ‘regular’ split does not produce more than 3 boxes). Of course, such an exponential worst case bound is expected in view of the NP-hardness of the global optimization problem (cf. Vavasis (1995)).

For the *MCS algorithm with local search* we obtain a stronger result if we make the obviously idealized assumption that the local search algorithm reaches a local minimizer after finitely many steps if it is started in its basin of attraction and if the function values at the nonglobal local minimizers are sufficiently separated from the global minimum.

THEOREM 2. *In addition to the assumptions of Theorem 1, assume that there is an $\varepsilon > 0$ such that $f(y) > f(\hat{x}) + \varepsilon$ for any nonglobal local minimizer $y$ and for any $y \in [u,v]$ with sufficiently large norm. Then there exist numbers $L$ and $S$ such that, for any $s_{\max} \geq L$, MCS with local search finds a global minimizer after at most $S$ sweeps.*

Note that the assumption on $f$ only excludes pathological optimization problems where a global optimizer is at infinity or where the set of nonglobal local optima has the global optimum as an accumulation point.

## 7. Numerical Results

### 7.1. Test Functions

Jones et al. (1993) gave an extensive comparison of their DIRECT method with various methods on seven standard test functions from Dixon and Szegö (1978) and two test functions from Yao (1989). Since our MCS algorithm is based on important insights from Jones et al. (1993), we first consider the same test set to evaluate the efficiency of MCS. For each test function, the dimensions and box bounds, used by Jones et al. (1993) but inadvertently omitted in Jones et al. (1993), are given in Table I. We thank Don Jones for providing us with the code of the test functions.

Table I. Dixon and Szegö (1978) functions: dimensions and box bounds.

| Label | Test function | Dimension | Default box bounds |
|---|---|---|---|
| S$m$ ($m = 5$, 7 or 10) | Shekel $m$ | 4 | $[0, 10]^4$ |
| H$n$ ($n = 3$ or 6) | Hartman $n$ | $n$ | $[0, 1]^n$ |
| GP | Goldstein–Price | 2 | $[-2, 2]^2$ |
| BR | Branin | 2 | $[-5, 10] \times [0, 15]$ |
| C6 | Six-hump camel | 2 | $[-3, 3] \times [-2, 2]$ |
| SHU | Shubert | 2 | $[-10, 10]^2$ |

Table II records the number of function calls needed for convergence. This is not the only way of assessing the quality of an algorithm, but it is an important one in the case of most real life applications, where function evaluations are expensive. All but the last four lines of Table II are taken from one of the tables of results of Jones et al. (1993); missing entries were not available from the literature. The first 11 algorithms already appeared in the 1978 anthology edited by Dixon and Szegö (1978) and are therefore somewhat old. The fourth last line contains results for the differential evolution algorithm DE by Storn and Price (1997); we used the MATLAB program `devec2.m` from `http://http.icsi.berkeley.edu/~storn/code.html` with the

Table II. Number of function calls for various methods compared to MCS.

| Method | S5 | S7 | S10 | H3 | H6 | GP | BR | C6 | SHU |
|---|---|---|---|---|---|---|---|---|---|
| Bremmerman | (a) | (a) | (a) | (a) | (a) | (a) | 250 | | |
| Mod. Bremmerman | (a) | (a) | (a) | (a) | 515 | 300 | 160 | | |
| Zilinskas | (a) | (a) | (a) | 8641 | | | 5129 | | |
| Gomulka–Branin | 5500 | 5020 | 4860 | | | | | | |
| Törn | 3679 | 3606 | 3874 | 2584 | 3447 | 2499 | 1558 | | |
| Gomulka–Törn | 6654 | 6084 | 6144 | | | | | | |
| Gomulka–V.M. | 7085 | 6684 | 7352 | 6766 | 11125 | 1495 | 1318 | | |
| Price | 3800 | 4900 | 4400 | 2400 | 7600 | 2500 | 1800 | | |
| Fagiuoli | 2514 | 2519 | 2518 | 513 | 2916 | 158 | 1600 | | |
| De Biase–Frontini | 620 | 788 | 1160 | 732 | 807 | 378 | 587 | | |
| Mockus | 1174 | 1279 | 1209 | 513 | 1232 | 362 | 189 | | |
| Bélisle et al. (1990) (b) | | | | 339 | 302 | 4728 | 1846 | | |
| Boender et al. (1982) | 567 | 624 | 755 | 235 | 462 | 398 | 235 | | |
| Snyman–Fatti (1987) | 845 | 799 | 920 | 365 | 517 | 474 | | 178 | |
| Kostrowicki–Piela (1991) | (c) | (c) | (c) | 200 | 200 | 120 | | 120 | |
| Yao (1989) | | | | | | | | 1132 | $< 6000$ |
| Perttunen (1990) | 516 | 371 | 250 | 264 | | 82 | 97 | 54 | 197 |
| Perttunen–Stuckman (1990) | 109 | 109 | 109 | 140 | 175 | 113 | 109 | 96 | (a) |
| Jones et al. (1993) | 155 | 145 | 145 | 199 | 571 | 191 | 195 | 285 | 2967 |
| Storn–Price (1997) (d) | 6400 | 6194 | 6251 | 476 | 7220 | 1018 | 1190 | 416 | 1371 |
| MCS (e) | $83^*$ | $129^*$ | $103^*$ | $79^*$ | $111^*$ | $81^*$ | $41^*$ | $42^*$ | $69^*$ |
| MCS (f)(d) | 582 | 633 | 595 | 131 | 113 | 94 | 51 | 37 | 566 |
| MCS (g)(e) | $196^*$ | $196^*$ | 330 | 128 | (c) | $194^*$ | $57^*$ | $44^*$ | $48^*$ |

(a) Method converged to a local minimum.
(b) Average evaluations when converges. For H6, converged only 70 % of time.
(c) Global minimum not found with less than 12 000 function calls.
(d) Average over 25 cases. For H6, average over 24 cases only; one case did not converge within 12 000 function values.
(e) An asterisk indicates that the first local optimization gave the global optimum.
(f) Perturbed box bounds
(g) Unconstrained problem.

default values for the control parameters. The number of function evaluations needed for convergence was averaged over 25 runs for each test function. In the case of Hartman6, one run did not converge after 12 000 function evaluations and we averaged only over the remaining 24 runs. The last three lines give results for MCS, first over the same bound constraints as in Jones et al. (1993), then averages over randomly perturbed box bounds, and finally for the unconstrained version. Details will be given below.

### 7.2. Termination

In the presentation of test results, methods are usually compared on the basis of their performance on problems with known solutions. The algorithm is terminated when a function value within some tolerance of the global minimum has been found, and we also adopt this strategy. However, in practical problems, one does not know the solution in advance and needs a criterion that tells the program when to stop searching for a better local minimizer. This criterion should be stringent enough that it does not waste too many function values after the global minimum has been found, but it should also be loose enough to ensure that in typical cases, the algorithm does not terminate before the global minimizer has been found.

Stochastic approaches to the design of suitable stopping criteria are surveyed in Section 6 of Boender and Romeijn (1995). One of the methods proposed there consists in stopping when the number $m$ of local searches done is larger than a function $N(w)$ of the number $w$ of different local minima found so far. The function $N(w)$ depends on the assumptions, and several specific implicit definitions of $N(w)$ are given in Boender and Romeijn (1995). This result is theoretically justified for the random multiple start method only but may serve as a guideline also for other methods that use local searches.

However, with MCS we try to do very few local optimizations only, and this reasoning appears inadequate. So far, we have not yet found a useful general purpose stopping criterion for MCS. For the purposes of the numerical tests reported here, the stopping criterion for MCS was taken as obtaining a relative error $< 0.01\,\%$ in the optimal objective function value (which happens to be nonzero always), i.e., $(f - f_{\text{glob}})/|f_{\text{glob}}| < 10^{-4}$, which was also the criterion used by Jones et al. (1993), Perttunen (1990), and Perttunen and Stuckman (1990). Since the DE algorithm of Storn and Price (1997) operates only at the global level, it takes a rather long time to find a minimum with high accuracy and therefore we used obtaining a relative error $< 1\,\%$ as stopping criterion. For the algorithms quoted in Jones et al. (1993), results based on the definition of convergence used by their authors are reported.

### 7.3. MCS Control Parameter Settings

We applied a MATLAB version of MCS with $s_{\max} = 5n + 10$, where $n$ is the dimension of the problem, and `smaxls` $= 15$ to the test functions and used a simple initialization list consisting of midpoint and boundary points, i.e.,

$$x_i^1 = u_i, \quad x_i^2 = (u_i + v_i)/2, \quad x_i^3 = v_i, \quad l_i = 2.$$

The limit on visits to Step 3 per local search was set to 50, and the parameter $\gamma$ in the stopping criterion for local optimization was taken as $\gamma = 10^{-18}$ (cf. Subsection 5.3). Note that all examples have been run with identical parameter settings, so that no tuning to the individual test problems was involved. For running MCS without local search, we would have to take a larger $s_{\max}$ (i.e., a larger number of levels) to reach the global minimum with the desired accuracy since then all of the local search has to be done by MCS.

### 7.4. Modified Bounds

We also investigated the stability of our results for MCS with respect to random perturbations of the box bounds. Instead of the default box bounds $[u, v]$ given in Table I, we employed the box bounds $[u', v']$ given by

$$u_i' = u_i + 0.5\eta(v_i - u_i), \quad v_i' = v_i + 0.5\eta(v_i - u_i), \quad i = 1, \ldots, n,$$

where $\eta$ is a random variable that is uniformly distributed in the interval $[-0.5, 0.5]$, but a value of $\eta$ was only accepted in a given problem if at least one of the global minimizers was in $[u', v']$. The results given in the second last line of Table II were taken as an average over 25 runs with different perturbed box bounds for each test function. For Hartman6, we obtained one outlier for which the algorithm had not found the global minimum after 12 000 function calls, and we report a result averaged over the 24 remaining runs.

Moreover, we applied MCS to the unconstrained optimization problem for the Dixon and Szegö (1978) test set and added the results to Table II. In this case, we cannot use an initialization list consisting of midpoint and boundary points any more. We used an initialization list consisting of $x_i^1 = -10$, $x_i^2 = 0$ and $x_i^3 = 10$ and again took $l_i = 2$ for $i = 1, \ldots, n$.

### 7.5. Discussion

The results show that MCS seems to be strongly competitive with existing algorithms in the case of problems with reasonable finite bound constraints. MCS with unperturbed box bounds wins in 8 of 9 test cases against every competing algorithm and is only beaten once by Perttunen–Stuckman for the remaining test function. MCS with perturbed box bounds still wins against all competing algorithms for four test functions. Only the results for Shekel's functions and Shubert's function seem to depend heavily on the choice of the box bounds, but they are comparable with the results of some other algorithms. Direct is also sensitive to perturbation of the box bounds (Jones, personal communication).

For unconstrained problems of dimension $n \geq 4$, the performance of MCS is less satisfactory. The reason is that in the exploration of an unbounded domain, it is easy to miss the region where the global minimum lies if one has already found a low-lying nonglobal minimizer. It seems that the MCS algorithm works reasonably well when the global minimizer can be localized reasonably well by the bound constraints, but not if the region containing the global minimizer is elusive.

In our Matlab version of MCS, the non-vectorizable loops lead to a significant overhead per function value. As the algorithm proceeds, the number of function values per sweep decreases, but the time spent per sweep does not become much shorter. This is due to the fact that, at later stages, the case that a box is processed without being split occurs more frequently. Since loops and conditional statements are executed in C much faster than in Matlab, a C implementation of the algorithm would drastically reduce the overhead.

### 7.6. Further Test Problems

The Dixon and Szegö (1978) test set has been criticized for containing mainly easy test problems. A more challenging test set was used in the first contest on evolutionary optimization (ICEO) at the ICEC'96 conference; cf. Storn and Price (1996). This test bed contains five problems, each in a 5-dimensional and a 10-dimensional version, and on these test functions, MCS showed some limitations. The names and default box bounds of the ICEO test functions are given in Table III, and the results are shown in Table IV. The first two lines in Table IV are results, taken from Storn and Price (1996), of two different versions of DE.

We applied MCS with three different choices of the initialization list to the ICEO test functions. MCS1 is the standard version with midpoints and boundary points. For MCS2, we took $x_i^1 = \frac{5}{6}u_i + \frac{1}{6}v_i$,

Table III. ICEO test functions and their box bounds.

| Problem | Name | Box bounds |
|---|---|---|
| 1 | Sphere model | $[-5, 5]^n$ |
| 2 | Griewank's function | $[-600, 600]^n$ |
| 3 | Shekel's foxholes | $[0, 10]^n$ |
| 4 | Michalewicz's function | $[0, \pi]^n$ |
| 5 | Langerman's function | $[0, 10]^n$ |

$x_i^2 = \frac{1}{2}(u_i + v_i)$, $x_i^3 = \frac{1}{6}u_i + \frac{5}{6}v_i$, $l_i = 2$, $i = 1, \ldots, n$, i.e., the points are uniformly spaced but do not include the boundary points.

For MCS3, we generated an initialization list with the aid of line searches. Starting with the absolutely smallest point in $[u, v]$, we made line searches with `gls` with `smaxls` $= 25$ and `nloc` $= 5$ along each coordinate, where the best point was taken as starting point for the next line search. The parameter `nloc` in `gls` determines how local or global the line search is since the algorithm tries to find up to `nloc` minima within `smaxls` function values. For the line searches in the local search method described in Section 5, `nloc` $= 1$ was taken (entirely local line search). For each coordinate $i$, all local minimizers found by the line searches were put into the initialization list, and if their number was less than three, they were supplemented with the values obtained from `gls` closest to $u_i$ and $v_i$.

Moreover, since some of the ICEO functions are considered to be hard problems, we also applied MCS with a larger number of levels, namely $s_{\max} = 10n$. Again we used the three different initialization lists defined above and added the results to Table IV as MCS4, MCS5 and MCS6, respectively.

For the ICEO functions, the uniformly spaced initialization list not containing any boundary points turned out to be most successful since these functions do not have any global minimizers at the boundary. Building an initialization list with the aid of line searches did not pay except for the easy ICEO1 function, where the minimizer was already found by the line searches, and the separable ICEO4 function. Moreover, taking a larger $s_{\max}$ yielded an improvement only for ICEO2.

Finally, we applied MCS to the testbed #1 used by Storn and Price (1997). The names and box bounds are shown in Table V, and their definition can be found in Storn and Price (1997). Problem 7 is a shifted version of ICEO2 for $n = 10$. Problems 8 and 9 have general constraints and were therefore not used here.

Table IV. Number of function values for the ICEO functions.

| | Problem 1 | | Problem 2 | | Problem 3 | | Problem 4 | | Problem 5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5D | 10D | 5D | 10D | 5D | 10D | 5D | 10D | 5D | 10D |
| DE1 | 736 | 1892 | 5765 | 13508 | 76210 | –[1] | 1877 | 10083 | 5308 | 44733 |
| DE2 | 463 | 1187 | 5157 | 16228 | 67380 | –[2] | 2551 | 18158 | 4814 | –[3] |
| MCS1 | 61 | 142 | – | – | 30050 | – | – | – | – | – |
| MCS2 | 62 | 142 | 1682 | 20904 | 1057 | 83713 | 25105 | – | – | – |
| MCS3 | 26 | 51 | – | – | 32901 | – | 1912 | – | – | – |
| MCS4 | 68 | 162 | 1271 | 15661 | 43377 | – | – | – | – | – |
| MCS5 | 68 | 162 | 1463 | 96903 | 1484 | – | 38826 | – | – | – |
| MCS6 | 26 | 51 | – | 46168 | 48424 | – | 1851 | – | – | – |

[1] 744250; [2] 203350; [3] 174006 function values.
A dash indicates that a global minimizer was not found after 100 000 function calls.

Table V. Testbed #1 of Storn and Price (1997), dimensions and box bounds.

| Problem | Name | Dimension $n$ | Box bounds |
|---|---|---|---|
| 1 | Sphere | 3 | $[-5.12, 5.12]^n$ |
| 2 | Rosenbrock | 2 | $[-2.048, 2.048]^n$ |
| 3 | Step | 5 | $[-5.12, 5.12]^n$ |
| 4 | Quartic with random noise | 30 | $[-1.28, 1.28]^n$ |
| 5 | Shekel's foxholes | 2 | $[-65.536, 65.536]^n$ |
| 6 | Corana's parabola | 4 | $[-1000, 1000]^n$ |
| 7 | Griewank's function | 10 | $[-400, 400]^n$ |
| 8 | Zimmerman's problem | 2 | constraints |
| 9 | Polynomial fit | 9, 17 | constraints |

The first three lines of Table VI are taken from Storn and Price (1997). ANM denotes the annealed Nelder & Mead strategy of Press et al. (1992) and ASA the Adaptive Simulated Annealing method by Ingber (1989; 1996).

For MCS, we used an initialization list consisting of midpoint and boundary points. However, for the functions where a known global optimizer happens to be among the points in the initialization list, a different initialization list with $L_i = 3$, $i = 1, \dots, n$, was chosen.

Table VI. Number of function values for the Storn and Price functions.

| | 1 | 2 | $3^1$ | $4^2$ | 5 | $6^1$ | 7 |
|---|---|---|---|---|---|---|---|
| ANM | 95 | 106 | 90258 | – | – | – | – |
| ASA | 397 | 11275 | 354 | 4812 | 1379 | 3581 | – |
| DE | 406 | 654 | 849 | 859 | 695 | 841 | 12752 |
| MCS | 10 | 111 | 45 | 673 | 3210 | 45 | – |

$^1$ discontinuous test function; $^2$ noisy test function.
A dash indicates that a global minimizer was not found after 100 000 function calls.

Problem 4 contains a random variable, and the result presented for MCS was averaged over 25 runs, where convergence was defined as reaching a point with function value $\leq 15$. Problem 1 has a quadratic objective function, hence is easy for MCS, and Problem 3 is easy for MCS since the objective function is monotonous.

## 8. Conclusions

The multilevel coordinate search algorithm MCS, presented in this paper, has excellent theoretical convergence properties if the function is continuous in the neighborhood of a global minimizer. In the current implementation, our test results show that MCS is strongly competitive with existing algorithms in the case of problems with reasonable finite bound constraints. In our comparison, MCS outperforms the competing algorithms almost always on the classical test problems set of Dixon and Szegö (1978) with bound constraints.

For unconstrained problems of dimension $n \geq 4$, the performance of MCS is less satisfactory, since in the exploration of an unbounded domain, it is easy to miss the region where the global minimum lies if one has already found a low-lying nonglobal minimizer. The same problem appears for some hard test problems with a huge number of local minima. However, whenever the global minimizer is found in these cases, the number of function evaluations is usually much smaller than for competing algorithms.

## Acknowledgements

The authors gratefully acknowledge partial support of this research by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung (FWF) under grant P11516-MAT.


## References


Bélisle, C. J. P., Romeijn, H. E. and Smith, R. L. (1990), Hide-and-Seek: a Simulated Annealing Algorithm for Global Optimization, Technical Report 90-25, Department of Industrial and Operations Engineering, University of Michigan.

Boender, C. G. E., Rinnoy Kan, A. H. G., Stougie, L. and Timmer, G. T. (1982), A Stochastic Method for Global Optimization, *Mathematical Programming* 22, 125–140.

Boender, C. G. E. and Romeijn, H. E. (1995), Stochastic Methods, in Horst, R. and Pardalos, P. M. (eds.), *Handbook of Global Optimization*, Kluwer, Dordrecht, 829–869.

Brent, R. P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, N. J.

Csendes, T. and Ratz, D. (1997), Subdivision Direction Selection in Interval Methods for Global Optimization, *SIAM J. on Numerical Analysis* 34, 922–938.

Dixon, L. C. W. and Szegö, G. P. (1978), The Global Optimization Problem: an Introduction, in Dixon, L. C. W. and Szegö, G. P. (eds.), *Towards Global Optimisation 2*, North-Holland, Amsterdam, 1–15.

Elster, C. and Neumaier, A. (1995), A Grid Algorithm for Bound Constrained Optimization of Noisy Functions, *IMA J. of Numerical Analysis* 15, 585–608.

Goertzel, B. (1992), Global Optimization by Multilevel Search, *J. of Optimization Theory and Applications* 75, 423–432.

Hansen, E. R. (1992), *Global Optimization Using Interval Analysis*, Dekker, New York.

Horst, R. and Tuy, H. (1996), *Global Optimization. Deterministic Approaches*, 3rd ed., Springer, Berlin.

Ingber, L. (1989), Very Fast Simulated Re-Annealing, *Mathematical and Computer Modelling* 12, 967–973.

Ingber, L. (1996), Adaptive Simulated Annealing (ASA): Lessons Learned, *Control and Cybernetics* 25, 33–54.

Jones, D. R., Perttunen, C. D. and Stuckman, B. E. (1993), Lipschitzian Optimization without the Lipschitz Constant, *J. of Optimization Theory and Applications* 79, 157–181.

Kostrowicki, J. and Piela, L. (1991), Diffusion Equation Method of Global Minimization: Performance on Standard Test Functions, *J. of Optimization Theory and Applications* 69, 269–284.

Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin.

Migdalas, A., Pardalos, P. M. and Värbrand, P. (1998), *Multilevel Optimization: Algorithms and Applications*, Kluwer, Dordrecht.

Nemhauser, G. L. and Wolsey, L. A. (1988), *Integer and Combinatorial Optimization*, Wiley, New York.

Perttunen, C. D. (1990), *Global Optimization Using Nonparametric Statistics*, PhD Thesis, University of Louisville.

Perttunen, C. D. and Stuckman, B. E. (1990), The Rank Transformation Applied to a Multiunivariate Method of Global Optimization, *IEEE Transactions on Systems, Man, and Cybernetics* 20, 1216–1220.

Pintér, J. D. (1996), *Global Optimization in Action*, Kluwer, Dordrecht.

Pintér, J. D. (1996), Continuous Global Optimization Software: a Brief Review, *Optima* 52, 1–8.

Powell, M. J. D. (1964), An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives, *Computer J.* 7, 155–162.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992), *Numerical Recipes in C*, 2nd ed., Cambridge University Press, Cambridge.

Ratz, D. and Csendes, T. (1995), On the Selection of Subdivision Directions in Interval Branch-and-Bound Methods for Global Optimization, *J. of Global Optimization* 7, 183–207.

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

Storn, R. and Price, K. (1996), Minimizing the Real Functions of the ICEC'96 Contest by Differential Evolution, in *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, IEEE Press, N.J., 842–844.

Storn, R. and Price, K. (1997), Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *J. of Global Optimization* 11, 341–359.

Vavasis, S. A. (1995), Complexity Issues in Global Optimization: a Survey, in Horst, R. and Pardalos, P. M. (eds.), *Handbook of Global Optimization*, Kluwer, Dordrecht, 27–41.

Vicente, L. N. and Calamai, P. H. (1994), Bilevel and Multilevel Programming: a Bibliography Review, *J. of Global Optimization* 5, 291–306.

Yao, Y. (1989), Dynamic Tunneling Algorithm for Global Optimization, *IEEE Transactions on Systems, Man, and Cybernetics* 19, 1222–1230.