Journal of Global Optimization (2005) 33: 61–82
DOI 10.1007/s10898-004-6097-6
© Springer 2005

Interval Branch and Bound with Local Sampling for Constrained Global Optimization

M. SUN[1] and A.W. JOHNSON[2]
[1]*Department of Mathematics, The University of Alabama, Tuscaloosa, AL 35487-0350, USA (e-mail: msun@gp.as.ua.edu)*
[2]*Computer-Based Honors Program, The University of Alabama, Tuscaloosa, AL 35487-0352, USA*

(Received 24 April 2003; accepted in revised form 5 November 2004)

Abstract. In this article, we introduce a global optimization algorithm that integrates the basic idea of interval branch and bound, and new local sampling strategies along with an efficient data structure. Also included in the algorithm are procedures that handle constraints. The algorithm is shown to be able to find all the global optimal solutions under mild conditions. It can be used to solve various optimization problems. The local sampling (even if done stochastically) is used only to speed up the convergence and does not affect the fact that a complete search is done. Results on several examples of various dimensions ranging from 1 to 100 are also presented to illustrate numerical performance of the algorithm along with comparison with another interval method without the new local sampling and several noninterval methods. The new algorithm is seen as the best performer among those tested for solving multi-dimensional problems.

Key words: Constraints, Global optimization, Interval branch and bound, Sampling

1. Introduction

One of the challenging mathematical problems of enormous theoretical and practical importance is how to find the globally optimal value of an objective function $f(x)$ and at least one global optimizer over a bounded multi-dimensional interval domain Ω in R^{d}, possibly subject to other equality and inequality constraints. That is,

$$\begin{aligned} &\text{minimize } f(x), \\ &\text{subject to } h(x)=0,\, g(x)\leqslant 0,\, x\in\Omega. \end{aligned} \tag{1}$$

Unlike local optimization problems, the global problem presents a number of difficult issues. First of all, there is no single verifiable sufficient condition for a globally optimal solution unless it is a very special case. Either a global behavior of $f(x)$ (for example, Lipschitz constant, cf. Clausen and Zilinskas, 2002) is used or the entire search domain is examined by global search algorithms. Usually two categories of search algorithms are thought

to be available. Deterministic algorithms for solving Problem (1) are generally based on the idea of branch and bound, which include interval methods (cf. Moore, 1979; Horst and Tuy, 1990) and cell exclusion methods (cf. Xu et al., 1997). Stochastic algorithms include probabilistic decision making that makes it possible to escape from locally optimal solutions in a probabilistic sense (cf. Kirkpatrick et al., 1983).

Interval branch and bound is one of the successful methods for solving the global problem. However stochastic search methods (such as the simulated annealing method and genetic algorithms) have been more popular choices because of their simplicity of implementation and relative quickness for reaching an approximate solution. But stochastic methods generally do not guarantee convergence to a global solution when a particular run ends. At most, they might guarantee convergence to a global solution in a probabilistic sense when the search process continues indefinitely. Even if a good approximate solution has been encountered during the search process, stochastic methods do not have reliable mechanism to detect it. This inevitably leads to additional and unnecessary computational efforts. Many stochastic search methods have been designed for solving unconstrained problems. Under the presence of constraints, their performance deteriorates even more and there are few theoretical justifications. Under the framework of interval branch and bound, a number of advantages are well known. (1) It guarantees convergence to all global solutions under weak assumptions, even in the presence of round-off errors. (2) It offers reliable stopping criteria so that the algorithm does not have to run longer than necessary. (3) It is numerically robust and handles round-off errors conveniently and effectively. (4) It is theoretically justifiable. (5) It handles constraints with relative ease and without jeopardizing theoretic justifications. Despite such attractive features of the interval method, most published reports on their applications seem to be generally limited to optimization problems in low dimensions (say, much less than 100 according to our recent survey). Obviously, there are two major concerns in solving large dimensional problems: large amount of memory space and slow speed of convergence. The number of subboxes to be processed in an interval method could potentially increase exponentially with the dimension of the domain. Inspired by such observations, we have investigated some new strategies associated with the interval branch and bound methodology both theoretically and numerically. This paper reports one new version of the interval algorithm that shows improvement both in memory space usage and in overall speed of convergence.

The rest of the paper is organized as follows. In Section 2, we review major features of the interval branch and bound. In Section 3, local sampling strategies are discussed. Our algorithm is presented in Section 4 along with theoretical convergence results. Two implementational strategies of the

interval method are also included. Numerical testing results are given in Section 5, followed by final comments and conclusions in Section 6.

2. Interval branch and bound

The interval branch and bound is a rather broad deterministic global optimization method. It uses interval arithmetics in the more general framework of branch and bound (cf. Moore, 1966 for earlier works on interval methods, Horst and Tuy, 1990 for more deterministic methods). The standard branch and bound method was originally introduced in Falk and Soland (1969) and Horst (1976), and more recently in Horst and Tuy (1990). It includes recursive refinement of partition of the search domain and underestimation of $f(x)$ over partitioned subdomains. Interval methods were initially developed around the same time (cf. Moore, 1966, 1979). Research on interval methods became a hot topic from late 1970s to early 1990s (cf. Alefeld and Herzberger, 1983; Ratscheck and Rokne, 1988; Neumaier, 1990; Hansen 1992) among many researchers in computer science, operations research, and mathematics. During that period of time, computers were becoming increasingly more popular and more powerful. Improved computer programming languages also helped promotion of interval methods. A solid foundation had been laid by the end of 1980s. Subsequent work certainly continues since 1990s (cf. Kearfott, 1996; Csallner, 2001; Clausen and Zilinskas, 2002; Van Voorhis, 2002). Our list of references is far from complete.

For ease of presentation, let $I(\Omega)$ be the set of all subintervals of Ω. Let f^* be the global minimum value of the objective function $f(x)$, and X^* the set of all global minimizers in Ω. As in the interval analysis literature, we use boxes and intervals interchangeably. A typical interval branch and bound method consists of these key ingredients.

1. The bounding principle: Use an inclusion function $F(X): I(\Omega) \to I(R)$, or any function that determines a lower bound of $f(x)$ over any X in $I(\Omega)$.
2. Subdivision of domain: The whole domain Ω is not searched uniformly for X^* due to concerns on computational efficiency. Instead, certain parts of Ω are searched more extensively than the others. The algorithm splits up Ω into subboxes where the bounding principle is applied. Usually, bisection is used recursively for this purpose.
3. A data structure that represents the remaining portion of the domain that still needs to be processed: Usually it is represented as a list of subboxes plus other attributes. We will refer to such a list as a list of nodes.
4. The branching principle (selection of a subbox for further processing): Usually the subbox with the lowest lower bound of the objective

function is selected for further processing (cf. Moore, 1966; Skelboe, 1974). Other selection criteria include the age or size of each subbox in the list (cf. Hansen, 1992).

5. Deletion conditions: To increase efficiency of the method, unwanted regions (where no global minimizer can be located) need to be identified and then deleted. The most commonly used deletion condition is the lower bound condition

$$f_{\text{best}} < \text{Lb}(F(X)), \tag{2}$$

where $\text{Lb}(F(X))$ is the lower bound of $f(x)$, and f_{best} is the currently known best value of the objective function. Later we will also use $\text{Ub}(F(X))$ as the upper bound of $f(x)$. The deletion condition (2) is similar to the so-called midpoint test (cf. Ichida and Fujii, 1979). When constraints are present in Problem (1), each processing subbox can be checked against the constraint satisfaction requirements. By using an inclusion function for each of the constraint functions, one can tell if a subbox is definitely infeasible or indeterminate. Thus the constraint verification adds an additional deletion condition. In this sense, the presence of constraints generally accelerates the deletion process of unwanted regions. That becomes an advantage of the interval method.
6. Termination criterion. Obviously, any interval algorithm stops if there are no more boxes to be processed.

There are a variety of implemented versions of the interval method. There are also several accelerating devices reported in the literature. Interval methods have been used for solving many different kinds of mathematical problems: optimization of functions, systems of linear and nonlinear equations, ordinary differential equations, partial differential equations, and optimal controls, just to name some.

3. Local sampling

Sampling of points in processing boxes is not necessary for early versions of the interval method. However, sampling is required to update f_{best} used in (2). The standard local sampling strategy is to use the midpoint of the local interval, resulting in the well-known midpoint test. Recently, Clausen and Zilinskas (2002) also used a vertex sampling strategy for a simplical branch and bound in global optimization of Lipschitzian functions. We introduce a variety of local sampling options for two (instead of one) major objectives: (a) to reduce the search space more quickly by identifying and deleting unwanted subregions as early as possible; (b) to provide a better upper bound f_{best} of f^*, which in turn would speed up the domain reduction process.

Our proposed sampling procedures include: (1) local optimization search sampling, (2) random sampling, (3) global fake sampling. Local optimization search sampling is favorable for finding a better upper bound of f^*. But it might come with a significant overhead of the computational cost. We note that both the midpoint sampling and the local optimization search sampling have been known for about two decades only for the purpose of upgrading f_{best}. We are extending their usage for another important objective, that is, to identify unwanted subregions. Thus we are taking advantage of their extra benefit without extra computational cost. Random sampling is simple both conceptually and computationally, and is also effective for both identifying unwanted subregions of high objective function values and providing a better upper bound of f^*. A small number of random samples are recommended. Its effectiveness is confirmed by our numerical results in Section 5. In any case, we will use the worst available sample point (e.g., with the highest objective function value) to determine the location of a subregion for early deletion since neighborhoods of the worst point are more likely to satisfy the box deletion conditions. Similarly, the best available sample point (e.g., with the lowest objective function value) is used to update f_{best}. In practice there are situations in which $f(x)$ is constant (or nearly so). In these cases, the quality of a solution could be measured in terms of constraint violations. It is emphasized that the local sampling (even if done stochastically) does not affect the fact that a complete search is done with a guaranteed global convergence. Our global fake sampling uses a finite initial value of f_{best}. This is only an estimated value of f^*. It is even not necessary to know the corresponding x_{best}. This "fake" or nominal f_{best} is used to accelerate the deletion process of subboxes. If this initial value is not too low, eventually it will be overridden by a real value of f_{best} for which x_{best} is indeed available. In that case, the use of the nominal f_{best} value does not effect the final outcome of the algorithm. It only accelerates the speed of convergence. However, if the initial value is set to be too low, a single run of the algorithm may result in no solution even when $X^* \neq \phi$ because boxes containing desired global solutions might have been mistakenly deleted due to a bad f_{best} value. If that actually happens, a new (larger) initial value of f_{best} can be estimated based on the previous "fake" value of f_{best} and the final real value of f_{best}. For example, the average value of the two might be a reasonable choice. The algorithm is then run again under the new f_{best} value. Of course, other sampling techniques can be incorporated just as easily.

In many cases of importance, a reasonable estimate of f^* may be readily available. Sometimes, even the exact value of f^* is also known. Consider an important class of optimization problems arising from solving a system

of linear or nonlinear equations

$$\phi(x) = 0. \tag{3}$$

Problem (3) is, if solvable, equivalent to

$$\min \|\phi(x)\|^2, \tag{4}$$

with $f^* = 0$, where $\|.\|$ is any norm. Thus in our method, the nominal value of f_{best} can be taken to be 0. If our interval method results in no solution to (4) under this fake sampling strategy, then the original problem (3) will have no solution either. Another important class of problems that would justify the fake sampling is the optimization problem for the least-squares data fitting. In this case, f^* should be a small positive number or zero. Consequently, the nominal value of f_{best} can be set to a small positive number. If only a fitted model with a desirable total fitting error tolerance is thought, then that tolerance value can be used as the nominal f_{best}. In case no optimal solution is found by our interval method, we would be able to conclude that no desirable fitted model exists. We also note that most noninterval methods are more efficient than interval methods for generating a rough estimate of f^*. Therefore, some noninterval methods may be used before any interval method is attempted. Thus, by the time this interval method is used, a fake value is already available.

When a subbox of the current processing box is deleted, usually 2d new boxes will be needed to represent the remaining region (cf. Figure 1 for illustration when $d = 2$) if only simple boxes (plus other attributes) are used

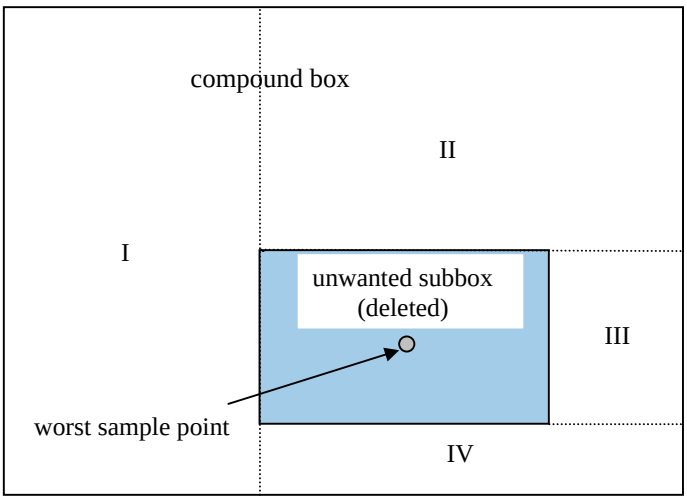

Figure 1. Compound box with 4 simple subboxes after deletion of a subbox.

in the data structure. However, we propose a more effective data structure that consists of up to two simple boxes plus other desired attributes. The first box is used to represent the current region (before deletion) as usual. Another box is allocated and it represents the unwanted region that can be deleted based on a local sampling procedure. In this way, we have saved memory space when d is larger than one, without sacrificing accuracy of information about the remaining processing region. Handling of this new data structure creates only an insignificant amount of CPU time overhead as shown by our numerical results in Section 5.

Even when we do not explicitly represent the compound region as shown in Figure 1 as a union of 2d simple box regions, the compound region is processed as if it were such a union. For example, when additional points are sampled in the compound box, we obtain those samples from some of its available subboxes. If we want to find a lower bound of the objective function value over the present compound box, we could find the lower bounds of the objective function values over those subboxes and then use the smallest of them as a desired lower bound. When explicit constraints are present, we add attributes for the compound region that would indicate which of the 2d subboxes violate at least one constraint. More generally, we could use appropriate attributes to indicate which of the 2d subboxes have been designated as inactive due to a constraint violation or satisfaction of any other deletion conditions. Such attributes are needed when the compound box needs to be branched or sampled.

4. Proposed algorithm and its convergence

Before the overall algorithm is described, we first outline major objects used in the algorithm as follows.

L_p = a primary list of nodes that represents the region to be searched. Each node consists of one box (X_1) or two boxes (X_1, X_2), plus attributes including a lower bound y_{lb} of $f(x)$ over this region, indicators IDS of deletion status (1 = deleted, 0 = remained) of its 2d subboxes, and indicators ICS of constraint satisfaction status (1 = satisfied, 0 = indeterminate). Let Y be the current processing box (simple or compound) and y as its y_{lb}.

L_s = a saved list of nodes that are not deleted but do not need to be further processed (i.e. inactive) according to some prescribed tolerances ($\varepsilon_{box}, \varepsilon_f$) listed below.

ε_{box} = a small box size threshold. Any active box X with size wid(X) less than ε_{box} will be moved from L_p to L_s.

ε_f = a small threshold of deviation of the objective function values. Any active box with the fluctuation of the objective function value less than ε_f will be stored into L_s as well.

f_{best} = the currently known best value of the objective function.

x_{best} = the currently known best ε − feasible solution.

X_{best} = the box that contains x_{best}.

$F(X) : I(\Omega) \to I(R)$, an inclusion function of $f(x)$.

$H_{\text{i}}(X) : I(\Omega) \to I(R)$, an inclusion function of each equality constraint function $h_{\text{i}}(x)$.

$G_{\text{j}}(X) : I(\Omega) \to I(R)$, an inclusion function of each inequality constraint function $g_{\text{j}}(x)$.

ε_{h} = a small threshold of absolute value of the equality constraint functions. Any active box X with $H_{\text{i}}(X)$ lying outside $[-\varepsilon_{\text{h}}, \varepsilon_{\text{h}}]$ will be designated as inactive due to violation of the equality constraint $h_{\text{i}}(x) = 0$.

ε_{g} = a small threshold of upper value of the inequality constraint functions. Any active box X with $G_{\text{j}}(X)$ lying outside $(-\infty, \varepsilon_{\text{g}}]$ will be considered as inactive due to violation of the inequality constraint $g_{\text{j}}(x) \leqslant 0$.

Procedure check_update_add_simple (X):

Begin check_update_add_simple():

Set y_{lb} for the box X.
Check: (1) deletion condition based on f_{best}; exit the procedure if X is deleted.
(2) deletion condition based on constraints; exit the procedure if X is deleted.
Update: If X cannot be deleted, sample its midpoint and update f_{best}, X_{best}, and x_{best} if the midpoint is ε-feasible and is better than x_{best}.
Add: If X cannot be deleted, add it to the end of L_{p}.

End check_update_add_simple().

Procedure check_update_add_compound (Y):

Begin check_update_add_compound():

a) For each of its available subboxes X (up to 2d of them), do the following.
Calculate y_{lb} for the box X.
Check: (1) deletion condition based on f_{best}; update Y's attributes if X is deleted.
(2) deletion condition based on constraints; update Y's attributes if X is deleted.
Update: If X cannot be deleted, sample its midpoint and update f_{best}, X_{best}, and x_{best} if the midpoint is ε-feasible and is better than x_{best}.
b) The surviving subbox with the lowest y_{lb} is deleted from Y and added to the end of L_{p}. Update Y's attributes accordingly. If the remaining Y is not empty, add it to the end of L_{p}.

End check_update_add_compound().

Proposed Algorithm:

Step 1. Initialization. Set $Y = \Omega$, and $y = y_{\text{lb}} = \text{Lb}(F(\Omega))$. Let $\text{L}_\text{s} = \text{L}_\text{p} = \phi$ (empty). Set parameters $\varepsilon_{\text{box}}, \varepsilon_\text{f}, \varepsilon_\text{h}, \varepsilon_\text{g}$ to small positive values (ε_g could be zero). Set all IDS and ICS components to 0. Set $f_{\text{best}} = \infty$, $X_{\text{best}} = \phi$, and $x_{\text{best}} = \phi$

Step 2. Update.

- 2a. If the current box Y is a simple box, apply a local sampling procedure to it. Update f_{best}, X_{best} and x_{best} if necessary. Identify and delete a subbox of this simple box if possible. Update its attributes accordingly.
- 2b. Check all the deletion conditions for the current box. If it can be discarded, do it and go to (2g).
- 2c. If the current node is still a simple box, split the box into two subboxes by a bisection along a direction perpendicular to an edge of the maximum length. Apply procedure check_update_add_simple(X) to both subboxes. If the current node is a compound box, apply Procedure check_update_add_compound (Y) instead.
- 2d. Discard some nodes from L_s based on the new f_{best}.
- 2e. Discard some nodes from L_p based on the new f_{best}.
- 2f. Move some nodes from L_p to L_s based on ε_{box} and ε_f.
- 2g. If $\text{L}_\text{p} = \phi$, stop.
- 2h. If some stopping criterion holds, stop.
- 2i. Select a new processing node from L_p and delete it from L_p. The selection can be based on the smallest attribute y_{lb}, the largest size, or the largest age, in a cyclic fashion. Set Y and its attributes. Repeat the updating step.

Step 3. Determination of final solutions. If $\text{x}_{\text{best}} \neq \phi$ (empty), it will be considered as an optimal solution. We may be able to get more optimal solutions from the survived lists L_p and L_s. In fact, for every box X from L_p and L_s, check its midpoint x. Consider x as another optimal solution if all the following conditions are satisfied. (1) x satisfies constraints within the prescribed tolerances. (2) $f(x) \leqslant \varepsilon_\text{f} + f_{\text{best}}$. (3) x is at least ε_{box} distance away from the current identified optimal solutions.

The presence of constraints generally accelerates the deletion process of unwanted regions as we pointed out in Section 2. However, constraints may also cause some headaches, especially the equality constraints. For example, the standard midpoints sampled in the algorithm may not be feasible at all. This prevents f_{best} from being updated. However, with our additional local

sampling procedures, f_{best} has a better chance to be updated. The relaxation of constraints also helps for this cause. Of course, other treatment options are also possible (e.g., Kearfott, 1996).

Many other strategies are available for further improvement. We only mention a couple of them below that have been used in our numerical experiment.

1. If a feasible solution is available before the algorithm is applied, the initial f_{best} would be set to the objective function value of that solution. The algorithm can be easily modified to keep track of best ε-feasible solutions sampled by the algorithm in case the initial fake value f_{best} is found to be totally wrong at the very end.
2. Stopping criteria. For practical considerations, the algorithm may be stopped when

   2a. #(f), The total number of calls of $f(x)$, or #(F), the total number of calls of its inclusion function $F(X)$, reaches a prescribed limit;
   2b. The total CPU time reaches a prescribed limit;
   2c. The combined size of the lists reaches a prescribed limit.

Like other branch and bound methods (e.g., Ratscheck and Rokne, 1988; Hansen, 1992), our new algorithm provides all desired solutions within the prescribed tolerances. It is emphasized that although interval arithmetic is used, the search is not perfectly rigorous because of the use of the tolerances. But we feel that such use of the tolerances greatly facilitates convergence. Since they are generally small, they do not affect the quality of final solutions for practical purposes. In some cases, they could be necessary computationally. For example, when $\varepsilon_{\text{h}} = 0$, a feasible point may never be found after any finite number of updating steps of the algorithm. We officially state our theoretical results in a theorem below. First, let LP_{n} be the sequence of the primary lists generated by the algorithm, and corresponding to the list LP_{n}, we denote

U_{n} = the union of the remaining search regions determined by all the boxes in LP_{n},
$(Y_{\text{n}}, y_{\text{n}})$ = the leading node for LP_{n}(plus other attributes),
f_{n} = the f_{best} for LP_{n},
$X^*(\varepsilon_{\text{h}}, \varepsilon_{\text{g}})$ = the set of optimizers under the relaxed constraints (or ε-feasible optimizers). $X^* = X^*(0, 0)$.
f_ε^* = the globally optimal objective function value under the relaxed constraints. $f^* = f_0^*$.

The saved list of nodes is used to reduce the computational cost by not splitting the boxes that are of a sufficiently small size or where the objective function varies very little. However, it is not included in our theorem below in order to obtain more concise theoretical convergence behaviors of the algorithm with more straightforward proofs.

THEOREM. Assumptions: 1. *All the inclusion functions* $\Phi(X)$ *used in the algorithm satisfy*

$$\text{wid}(\Phi(X)) \to 0 \text{ as } \text{wid}(X) \to 0.$$

2. $\varepsilon_{\text{h}} > 0,\ \varepsilon_g \geqslant 0.$
3. *There is at least one* x_ε^* *in* $X^*(\varepsilon_{\text{h}}, \varepsilon_{\text{g}})$ *satisfying* $|h_{\text{i}}(x_\varepsilon^*)| < \varepsilon_{\text{h}}$ *and* $g_{\text{j}}(x_\varepsilon^*) < \varepsilon_{\text{g}}$ *for all i and j.*
4. *The algorithm does not terminate after a finite number of updating iterations.*
5. *The saved list of nodes is not used.*

Conclusions: 1. *wid*$(Z_{\text{n}}) \to 0$ *for any box* Z_{n} *in* LP_{n}.

2. $y_{\text{n}} \leqslant f_\varepsilon^*$ *and* $y_{\text{n}} \to f_\varepsilon^*$.
3. $f_{\text{n}} \downarrow f_\varepsilon^*$.
4. *The sequence* $\{U_{\text{n}}\}$ *is nested and* $U_{\text{n}} \to X^*(\varepsilon_{\text{h}}, \varepsilon_{\text{g}})$ *under the Hausdorff distance.*

Proof. In our proof, we frequently encounter sequences and their subsequences. For simplicity of notation, we often use the same notation for a sequence and its subsequences. However we will specify when a new subsequence is introduced.

1. According to our alternating selection strategy, every surviving box in the processing list will get processed at least once for a while. Hence we get

$$\text{wid}(Z_{\text{n}}) \to 0 \text{ for any box } Z_{\text{n}} \text{ in } \text{LP}_{\text{n}} \tag{5}$$

   as for the standard Hansen's algorithm.
2. It is fairly obvious that $y_{\text{n}} \leqslant f_\varepsilon$* for any n since only some unwanted regions have been discarded. Now we want to show

$$y_{\text{n}} \to f_\varepsilon^*.$$

   Since $\{y_{\text{n}}\}$ is a bounded sequence, it suffices to show that every limit point of y_{n} is f_ε*. Suppose not. Then there would be a convergent subsequence (still denoted by itself for simplicity)

$$y_{\text{n}} \to f^{\#} \text{ for some } f^{\#} < f_\varepsilon^*. \tag{6}$$

   Under this circumstance, we now show that $f_\varepsilon^* \in F(Y_{\text{n}})$ for large enough n, from which we get

$$y_{\text{n}} = \text{Lb}(F(Y_{\text{n}})) \to f_\varepsilon^* \text{ since } \text{wid}(F(Y_{\text{n}})) \to 0,$$

contradicting (6). Suppose $f_\varepsilon^* \notin F(Y_\mathrm{n})$ for a subsequence of $\{\mathrm{n}\}$, still denoted by itself again for notational simplicity. Then

$$y_\mathrm{n} = \mathrm{Lb}(F(Y_\mathrm{n})) \leqslant \mathrm{Ub}(F(Y_\mathrm{n})) \leqslant f^\# < f_\varepsilon^*, \tag{7}$$

possibly for another $f^\#$. Thus every point in Y_n is not ε-feasible.
Claim: Under condition (7), $\exists \delta > 0$ and a subsequence of $\{\mathrm{n}\}$ such that for all large enough n in the new subsequence, either

$$\exists i = i_\mathrm{n} \text{ with } |h_\mathrm{i}(x)| > \varepsilon_\mathrm{h} + \delta \quad \text{for all } x \in Y_\mathrm{n}, \tag{8a}$$

or

$$\exists j = j_\mathrm{n} \text{ with } g_\mathrm{j}(x) > \varepsilon_\mathrm{g} + \delta \quad \text{for all } x \in Y_\mathrm{n}. \tag{8b}$$

Suppose not. We first choose a subsequence of $\{Y_\mathrm{n}\}$ converging to a single point. This is possible due to (5). Then

$$\forall i, \exists \delta(k,i) \downarrow 0, \mathrm{n}(k,i) \uparrow \infty (\text{as } k \uparrow \infty), \text{ and } x^{(\mathrm{n}(k,i),i)} \in Y_{\mathrm{n}(k,i)},$$
$$\text{such that } |h_\mathrm{i}(x^{(\mathrm{n}(k,i),i)})| \leqslant \varepsilon_\mathrm{h} + \delta(k,i);$$

and

$$\forall j, \exists \delta(k,j) \downarrow 0, n(k,j) \uparrow \infty (\text{as } k \uparrow \infty), \text{ and } x^{(\mathrm{n}(k,j),j)} \in Y_{n(k,j)},$$
$$\text{such that } g_\mathrm{j}(x^{(\mathrm{n}(k,j),j)}) \leqslant \varepsilon_\mathrm{g} + \delta(k,j).$$

By compactness of each box,
$\forall i$, $\exists$ a convergent subsequence (still denoted by the original sequence)

$$x^{(\mathrm{n}(k,i),i)} \to x^{(\mathrm{i})} \text{ with } |h_\mathrm{i}(x^{(\mathrm{i})})| \leqslant \varepsilon_\mathrm{h};$$

$\forall j$, $\exists$ a convergent subsequence (still denoted by the original sequence)

$$x^{(\mathrm{n}(k,j),j)} \to x^{(j)} \text{ with } g_\mathrm{j}(x^{(j)}) \leqslant \varepsilon_\mathrm{g}.$$

Since the subsequence Y_n converges to a single point, $x^{(i)} = x^{(j)} = x'$ for some x' in Ω and all (i, j). This x' is therefore ε-feasible, that is

$$|h_i(x')| \leqslant \varepsilon_\mathrm{h}, g_\mathrm{j}(x') \leqslant \varepsilon_\mathrm{g} \text{ for all } i \text{ and } j.$$

But this is impossible because of (7).

Assumption 1 and inequalities (8) imply that $\mathrm{Y_n}$ should have been deleted from the processing list by our algorithm due to the constraint violation. This shows $f_\varepsilon^* \in F(Y_\mathrm{n})$ for large enough n under condition (6).

3. The monotonicity of f_n is fairly clear because f_n represents the currently available best objective function value at updating iteration n. It remains to show f_ε^* as its limit. In fact, it suffices to show that a subsequence of $\{f_n\}$ converges to f_ε^*. Every list LP_n should include a box Z_n that contains x_ε^* from Assumption 3 since the optimal solution is never deleted. By Assumptions 1–3 and Conclusion 1, we have for large n

$$H_i(Z_n) \subseteq [-\varepsilon_h, \varepsilon_h] \text{ and } \mathrm{Ub}(G_j(Z_n)) \leqslant \varepsilon_g \text{ for all } i \text{ and } j.$$

For such a box, we also have

$$f_\varepsilon^* \leqslant f_n \leqslant f(x_{n(\text{local search})}) \leqslant f(x_\varepsilon^*) + \mathrm{wid}(F(Z_n)),$$

where $x_{n(\text{local search})}$ is the best ε-feasible point identified by our local sampling procedure over the box Z_n. Our desired convergence result follows from $\mathrm{wid}(F(Z_n)) \to 0$.

4. It suffices to show $\cap U_n \subseteq X^*(\varepsilon_h, \varepsilon_g)$ since only some unwanted regions might have been discarded. Again we will try to get a contradiction if we assume that, for some $x_0 \notin X^*(\varepsilon_h, \varepsilon_g)$, $x_0 \in \cap U_n$. Every list LP_n includes a box Z_n that contains this x_0. Because of Conclusion 1, we have $Z_n \to x_0$ and consequently, $F(Z_n) \to f(x_0) > f_\varepsilon^*$ due to continuity of $f(x)$. In view of Conclusion 3, we would have

$$\mathrm{Lb}(F(Z_n)) > f_n \text{ for large enough n.}$$

Thus, such a box Z_n should have been deleted by the algorithm, which implies the desired contradiction. This completes the proof of the whole theorem. □

COROLLARY. *In absence of constraints, the conclusions remain true for* $\varepsilon = 0$ *(that is,* $X^*(\varepsilon_h, \varepsilon_g) = X^*$ *and* $f_\varepsilon^* = f^*$*) without Assumptions 2–3.*

5. Numerical results

To test performance of the new algorithm (denoted by IVL), we have used several examples with or without constraints. Preliminary results on six examples are presented below. Their dimensions vary from 1 to 100. Most of these examples have been widely used by other people for testing their new optimization algorithms (e.g., Floudas and Pardalos, 1990). We have also used the corresponding interval algorithm with just the midpoint sampling (IVM) along with a conventional simple node structure,

and four (noninterval) stochastic methods to solve the same test problems for comparison. The four stochastic methods are a simulated annealing algorithm (SA, cf. Kirkpatrick et al., 1983), a tree annealing algorithm (TA, cf. Sun, 2002, as a newer variant of SA), a genetic algorithm (GA, cf. Goldberg, 1989), and a complete random search (RS). The noninterval methods incorporate the constraints into the objective function through a well-calibrated penalty term. The comparison results presented below are by no means comprehensive and 100% realistic. In fact, we find it difficult to do a very realistic comparison since each algorithm has a number of its own control parameters that could affect its individual performance. It is difficult and probably impossible to find "equivalent" sets of all control parameters for two different algorithms. However, for each example, we have used the same set of the shared parameters and the same initial solutions for all algorithms. To further increase reliability of the test results on stochastic methods, each example is run 40 times using randomly selected initial solutions and seeds for the random number generator while the other parameters are fixed. The mean and standard deviation information will be summarized. According to the statistics theory, statistically reasonable conclusions could be drawn using 40 independent samples. Thus, the results below could still provide some idea of effectiveness of the new algorithm.

As usual, those noninterval algorithms don't have precise stopping conditions that would ensure immediate exit once a globally optimal solution is found unless Problem (1) is special enough. But within any of the six algorithms we never intentionally used any special information of objective functions or constraint functions although some of them seem fairly special. Therefore, we have to use some brutal stopping conditions such as these: (1) the number of calls of the objective function reaches a prescribed limit; (2) the total computational time exceeds a specified limit; (3) the number of generations is too big for GA; (4) there is no improvement of f_{best} over a fixed number of iterations. Such stopping conditions may result in a premature termination of any algorithm. They may also result in some wastes of unnecessary iterations after reaching an optimal solution. Unfortunately, these are just some of the pitiful features of many noninterval algorithms.

As stated in Section 2, an inclusion function is usually required for solving each optimization problem by using an interval method. Inclusion functions are not unique. In fact, there are several well-known approaches for constructing inclusion functions (cf. Ratscheck and Rokne, 1984). (1) If the noninterval function can be described in some programming language as an explicit expression without use of logical or conditional statements, then the natural interval extension of it can be defined easily. (2) If an inclusion function of the gradient or generalized gradient can be

found, the meanvalue form can be used as an inclusion function. (3) If an inclusion function of the Hessian matrix can be found, the 2nd-order Taylor form can be used instead. In the examples presented below, we use the natural interval extension whenever possible unless it is specified explicitly.

Test results for each example are summarized in a separate table. The first column of each table shows different algorithms used for that example. The second column indicates the number of runs under the same set of parameters but with random initial trial solutions and random seed values for the random number generator that is needed for all of our (noninterval) stochastic methods used for comparison. The next column lists the mean and standard deviation of the final solutions (represented by f_{best}) obtained from different runs. The standard deviation is skipped if there is only one run for an interval algorithm. The forth column lists the mean and standard deviation of $\#(f)$, the total number of calls of $f(x)$ for each run. For interval algorithms, we have split that number into two parts, $\#(f)$, the total number of calls of $f(x)$, plus $\#(F)$, the total number of calls of its inclusion function $F(X)$. Usually, a call of $F(X)$ is computationally more expensive than a call of $f(x)$ (in fact, at least several times more expensive). Our test results show that in some cases, algorithm IVL is much more efficient than IVM because it takes a significantly smaller number of $F(X)$ calls although it usually requires more calls of $f(x)$ due to the new local sampling. The next column displays the mean and standard deviation of the total CPU time per run of each method that would include real CPU time plus all the auxiliary times such as I\O times. However, we didn't keep the CPU time if it takes less than one millisecond. But, the mean and standard deviation of CPU times displayed may be less than one millisecond since they are literally calculated from the individual CPU time values. The final column shows the number of approximate global minimizers actually identified by the algorithm after the specified number of runs followed by, within the parentheses, the number of different optimal solutions identified by those runs. We have assumed that a single run of any of the four stochastic algorithms can identify at most one approximate global minimizer. Although all the algorithms might find only one optimal solution for a certain example, only the interval methods ensure us that this is indeed the only solution of that example. Even with 40 independent runs, the noninterval algorithms only tell us with a large statistic confidence that there is no other global solution of this example. In fact, some examples below would convince us that this statistic conclusion might actually be false sometimes. All of the test results have been generated by a Pentium IV 1.6 Ghz PC. As we present test results of each example, we will also

Table I. Summary of test results for Example 1

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	0.418958, 0.0081727	55, 2	7.5e−3s, 4.2e−4s	39(1)
GA	40	0.398692, 4.8e−9	38235, 237	0.84s, 5.0e−3s	40(1)
TA	40	0.398694, 7.09e−7	47572, 0	0.26s, 1.7e−3s	40(1)
RS	40	0.398692, 0	100000, 0	0.75s, 2.0e−3s	40(1)
IVM	1	0.398693	116+165	0.01s	1
IVL	1	0.398693	1016+179	0.02s	1

make some important observations and offer brief discussions about the results.

EXAMPLE 1. Minimize $f(x) = 5.0 + \sin(x) + \sin(10x/3) + \log(x) - 0.84x$, over [2.7, 7.5]. Summary of the test results is in Table 1. SA tends to exit prematurely sometimes. As a result, it used a much smaller number of calls of $f(x)$. All the other algorithms successfully identified the unique globally optimal solution after every run at the expense of more CPU time. RS took the most f-calls because the limit on f-calls is the stopping condition. The two interval algorithms took a small number of f-calls, while still providing a reliable final solution within a compatible amount of CPU time. No advantage of the new algorithm IVL is seen over its counterpart IVM for this one-dimensional example. In fact, its overhead of computational effort seems dominant. This is, however, not beyond our expectations.

EXAMPLE 2. (cf. Yao et al. 1999). Minimize a six hump camel back function

$$f(x_1, x_2) = 1.03163 + 4.0x_1^2 - 2.1x_1^4 + x_1^6/3.0 + x_1x_2 - 4.0(1.0 - x_2^2)x_2^2$$

over $[-5, 5] \times [-5, 5]$. Summary of its test results is in Table 2. This example has two globally optimal solutions. Only the interval algorithms successfully identified both optimal solutions on a single run. The new

Table II. Summary of test results for Example 2

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	357.345, 77.5478	104, 4	2.5e−3s, 6.8e−4s	5(2)
GA	40	1.935e−6, 1.197e−7	61106, 2915	1.5s, 0.13s	40(2)
TA	40	0.0045, 0.00154	104191, 0	0.44s, 4.7e−3s	40(2)
RS	40	0.00205634, 0	100000, 0	1.3s, 2.5e−3s	40(1)
IVM	1	1.79869e−006	22531+32579	3m18.2s	2
IVL	1	2.22992e−006	50007+9068	15.1s	2

interval algorithm IVL clearly outperforms the regular version IVM in terms of the overall CPU time counts. The overhead of computational effort of IVL is no longer dominant for this two-dimensional example as we expected. The savings of CPU time under IVL come from the reduction of $\#(F)$ while keeping $\#(f) + \#(F)$ roughly competitive. That is our major expected advantage of the new method. Again, SA shows a much worse average value of f_{best} due to many runs with premature exits. However, we noticed that a few best runs of SA also reached a nearly optimal solution. Another important observation should be made regarding RS. Although each of 40 independent runs finds an optimal solution. But they actually reached the same global solution. The other global solution is completely missed. This also tells us that using multiple starting points for a noninterval method is not a sure way to get all the global solutions. In fact, there is no way to determine how many starting points are needed to get all the global solutions.

EXAMPLE 3. (cf. Sun, 2002). Minimize $f(x) = \| \max\{A_1 x - b_1,\ A_2 x - b_2\} \|_1$ over $[-100, 100] \times [-100, 100] \times [-100, 100]$, where

$$A_1 = \begin{bmatrix} 1 & 0 & -0.5 \\ -0.5 & 1 & 0 \\ 0 & -0.5 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} 1 & -0.5 & 0 \\ -2/3 & 1 & -1/5 \\ -1/4 & -1/3 & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.$$

Notice that this function is not smooth. This problem is equivalent to the discretized version of the Hamilton–Jacobi–Bellman equation associated with an optimal control problem.

LEMMA. *Consider*

$$F(X) = \Sigma_{i=1,2,3}\ AbsIvl\ (MaxIvl((A_1 X - b_1)_i, (A_2 X - b_2)_i)),$$

where AbsIvl(A) is an inclusion function of $|x|$, and MaxIvl(A, B) is an inclusion function of $\max\{x, y\}$. Then $F(X)$ is an inclusion function of $f(x)$.

Table 3 shows that for a single run, only the interval methods find the global solution accurately. RS fails to get any decent solution at all after 40 independent runs. Thus the interval methods are more reliable. This example also shows more computational cost of IVL than IVM partially because IVL reached a more accurate solution while IVM exited earlier.

Table III. Summary of test results for Example 3

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	6.98973, 1.61326	414, 31	9.8e−3s, 9.7e−4s	5(1)
GA	40	0.210567, 0.02329	104779, 2725	2.9s, 0.15s	35(1)
TA	40	4.17542, 0.3598	116851, 0	0.57s, 9.0e−3s	2(1)
RS	40	3.03519, 7.0e−17	100000, 0	1.8s, 1.8e−3s	0(0)
IVM	1	0.000165623	676+811	0.060s	1
IVL	1	4.631e−005	18463+3900	0.91s	1

EXAMPLE 4. (cf. Floudas and Pardalos, 1990)

$$\begin{aligned} \text{Minimize } & f(x) = x_1^{0.6} + x_2^{0.6} - 6x_1 - 4x_3 + 3x_4, \\ \text{subject to } & x_1 - 1/3x_2 + x_3 = 0, \\ & x_1 + 2x_3 \leqslant 4, \\ & x_2 + 2x_4 \leqslant 4, \end{aligned}$$

over $[0, 3] \times [0, 5] \times [0, 1] \times [0, 2]$. This is a problem with equality and inequality constraints mixed. Summary of its test results is in Table 4. The table does not show $\#(H)$ and $\#(G)$. However, our output files show that they are usually of the same order of magnitude as $\#(f)$.

Constraint verification is internally incorporated into the interval methods. But the noninterval methods enforce the constraints through a penalty function (thus $\#(g) = \#(h) = \#(f)$). The penalty function method comes with a penalty coefficient whose value would effect the performance of those noninterval methods. In all our test results with the penalty function method, we only present results under one properly selected penalty coefficient value. Additional runs might have been done to calibrate its value. Even under such favorable conditions, the noninterval methods still do not outperform the interval methods.

This example shows a significant amount of increase in CPU time consumption by the interval methods. One major reason is due to existence of equality constraints. It is well know that equality constraints may make it harder for interval methods to converge quickly. Another reason is that the numerical results on the noninterval methods are based on a carefully selected penalty

Table IV. Summary of test results for Example 4

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	−1.20825, 0.22877	288, 16	5.8e−3s, 7.9e−4s	6(1)
GA	40	−4.26767, 0.01266	98187, 4363	3.2s, 0.21s	40(1)
TA	40	−3.22012, 0.11016	93447,0	1.4s, 0.017s	28(1)
RS	40	−3.27683, 7.0217e−17	200000,0	5.2s, 6.3e−3s	40(1)
IVM	1	−4.45083	38805+100001	33m52s	1
IVL	1	−4.4828	104141+18053	59s	1

coefficient. Nevertheless, the interval methods offer a better final solution. Again, IVL clearly shows a much better performance than IVM.

EXAMPLE 5. (cf. Michalewicz, 1996). Minimize

$$
\begin{aligned}
f(x) &= 5(x_1+x_2+x_3+x_4) - 5(x_1^2+x_2^2+x_3^2+x_4^2) \\
&\quad -(x_5+x_6+\cdots+x_{12}+x_{13}), \\
\text{subject to } & 2x_1+2x_2+x_{10}+x_{11} \leqslant 10, \\
& 2x_1+2x_3+x_{10}+x_{12} \leqslant 10, \\
& 2x_2+2x_3+x_{11}+x_{12} \leqslant 10, \\
& -8x_1+x_{10} \leqslant 0, \\
& -8x_2+x_{11} \leqslant 0, \\
& -8x_3+x_{12} \leqslant 0, \\
& -2x_4-x_5+x_{10} \leqslant 0, \\
& -2x_6-x_7+x_{11} \leqslant 0, \\
& -2x_8-x_9+x_{12} \leqslant 0, \\
& 0 \leqslant x_i \leqslant 1,\ i=1,2,3,4,5,6,7,8,9,13, \\
& 0 \leqslant x_i \leqslant 8,\ i=10,11,12.
\end{aligned}
$$

Summary of the test results for Example 5 is in Table 5. The number of calls of $G(X)$ for interval algorithms IVM and IVL is, respectively, 255723 and 391[35]. This significant difference leads to a much greater amount of CPU time savings for IVL.

EXAMPLE 6. (cf. Yao et al. 1999). Minimize a generalized Rastrigin's function

$$
f(x) = \sum_{i=1}^{100} [x_i^2 - 10\cos(2\pi x_i) + 10] \text{ over } -5.12 \leqslant x_i \leqslant 5.12, \quad i = 1, \ldots, 100.
$$

This is a higher-dimensional example. Its lower-dimensional versions are usually adopted as test examples by many researchers. Summary of its test results is in Table 6. Once again, the interval methods remain reliable in

Table V. Summary of test results for Example 5

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	−15.8298, 0.35577	2313, 130	0.047s, 2.8e−3s	33(1)
GA	40	−16.8466, 0.04956	114803, 7673	6.4s, 0.45s	40(1)
TA	40	−8.9684, 0.167684	133155, 0	2.5s, 0.014s	1(1)
RS	40	−8.36159, 0	200000, 0	14.7s, 0.088s	0(0)
IVM	1	−16.8127	24+100001	48m40s	1
IVL	1	−16.9881	200001+18974	1m11s	1

Table VI. Summary of test results for Example 6

alg	#(run)	f-best (mean,std)	#(f-call) mean,std	time(h,m,s) mean,std	#(x^*)
SA	40	18.2559, 10.5652	53250, 3470	1.9s, 0.18s	29(1)
GA	40	761.623, 11.6514	269215, 14240	0.48m, 0.079m	0(0)
TA	40	929.68, 2.93471	221982,0	13.8s, 0.13s	0(0)
RS	40	1431.82, 3.6e−14	400000, 0	3m41s, 0.24s	0(0)
IVM	1	0.000668513	198811+200001	8h8m27s	1
IVL	1	0.000710523	200000+36566	17m27s	1

finding the global solution and the new interval method IVL is clearly the best.

With or without constraints, IVL is consistently shown to be better than IVM for higher-dimensional problems, thanks to introduction of adequate local sampling. This new strategy works as an effective accelerating device for the interval method for two major reasons. (1) It accelerates the deletion process by improving f_{best} more quickly. (2) It finds unwanted regions more aggressively and more accurately than the standard interval method so that a much smaller number of boxes would have to be processed. These two major objectives are achieved in our interval method with assistance of a new data structure, all at only a small fraction of computational overhead.

6. Final comments

Based on the test results presented earlier, a number of observations can be made.

(1) The major advantages of the interval algorithms over noninterval algorithms include: (a) They are more reliable. Example 6 above shows significant decay of performance of all four noninterval algorithms. But the interval algorithms remain effective. (b) They are robust, not much affected by round-off errors. This is a useful feature especially for optimization problems involving uncertainty. (c) They offer all the global solutions by a single run. Even multiple runs of noninterval algorithms with different starting points cannot guarantee finding all those solutions. By the way, out TA is capable of identifying more optimal or nearly optimal solutions than most other noninterval methods as explained in Sun (2002). However it still does not offer any guarantee as the interval methods do.

(2) One noticeable disadvantage of the interval methods is that they generally require more memory and CPU time than the noninterval algorithms for a single run. Such a problem may become more visible when the optimization problem is of a large dimension. It may also occur when the structure of the optimization problem is too simple or too complicated.

In many practical applications, finding a desirable solution with a guaranteed accuracy is important at the cost of any reasonable CPU time. Interval methods can offer such desirable solutions.

(3) Effectiveness of the new local sampling strategy in the interval method: Interval algorithm IVL with the new local sampling strategy consistently performs at least as competitively well as the interval method with only the standard midpoint sampling strategy. In higher-dimensional cases, IVL seems to be significantly better than IVM.

Not every feature of the new algorithm has been extensively tested yet. We are constantly improving it as more tests are done. Nevertheless, the existing results are encouraging and show a good promise. We hope that this paper would stimulate further research efforts in this area.

Acknowledgement

The authors are grateful for the comments by two referees on an earlier version of this paper.

References

1. Alefeld, G. and Herzberger, J. (1983), *Introduction to Interval Computations*, Academic Press, New York, NY.
2. Clausen, J. and Zilinskas, A. (2002), Subdivision, sampling, and initialization strategies for simplical branch and bound in global optimization, *Computers & Mathematics with Applications* 44, 943–955.
3. Csallner, A.E. (2001), Lipschitz continuity and the termination of interval methods for global optimization, *Computers & Mathematics with Applications* 42, 1035–1042.
4. Falk, J.E. and Soland, R.M. (1969), An algorithm for separable nonconvex programming problems, *Management Science* 15, 550–569.
5. Floudas, C.A. and Pardalos, P.M. (1990), *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, Berlin Heidelberg.
6. Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.
7. Hansen, E.R. (1992), *Global Optimization using Interval Analysis*, Marcel Dekker, NY.
8. Horst, R. (1976), An algorithm for nonconvex programming problems, *Mathematical Programming* 10, 312–321.
9. Horst R. and Tuy, H. (1990), *Global Optimization, Deterministic Approaches*, Springer-Verlag, Berlin.
10. Ichida, K. and Fujii, Y. (1979), An interval arithmetic method for global optimization, *Computing* 23, 85–97.
11. Kearfott, R.B. (1996), A review of techniques in the verified solution of constrained global optimization problems. In: *Applications of Interval Computations* (El Paso, TX, 1995), Appl. Optim., vol. 3, Kluwer Acad. Publ., Dordrecht, 23–59.
12. Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer-Verlag, Berlin.

13. Kirkpatrick, S., Gelatt Jr. C.D. and Vecchi, M.P. (1983), Optimization by simulated annealing, *Science* 220, 671–680.
14. Moore, R.E. (1966), *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ.
15. Moore, R.E. (1979), *Methods and Applications of Interval Analysis*, SIAM Publication, Philadelphia, Pennsylvania.
16. Neumaier, A. (1990), *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, U.K.
17. Ratscheck, H. and Rokne, J. (1984), *Computer Methods for the Range of Functions*, Ellis Horwood Limited, Chichester.
18. Ratscheck, H. and Rokne, J. (1988), *New Computer Methods for Global Optimization*, Wiley, New York, NY.
19. Skelboe, S. (1974), Computation of rational interval functions, *BIT* 14, 87–95.
20. Sun, M. (2002), Tree annealing for constrained optimization, In: *Proceedings of 34th IEEE Southeastern Symposium on System Theory*, Huntsville, AL, pp. 412–416.
21. Van Voorhis, T. (2002), A global optimization algorithm using Lagrangian underestimates and the interval Newton method, *Journal of Global Optimization* 24, 349–370.
22. Xu, Z.B., Zhang, J.S., and Leung, Y.W. (1997), A general CDC formulation for specializing the cell exclusion algorithms of finding all zeros of vector functions, *Applied Mathematics and Computation* 86, 235–259.
23. Yao, X., Liu, Y. and Lin, G. (1999), Evolutionary programming made faster, *IEEE Transactions on Evolutionay Computation* 3, 82–102.