Noname manuscript No.
(will be inserted by the editor)

A Metaheuristic Methodology based on the Limitation of the Memory of Interval Branch and Bound Algorithms

Jordan Ninin, Frédéric Messine

the date of receipt and acceptance should be inserted later

Abstract Branch and Bound Algorithms based on Interval Arithmetic permit to solve exactly continuous (as well as mixed) non-linear and non-convex global optimization problems. However, their intrinsic exponential time-complexities do not make it possible to solve some quite large problems. The idea proposed in this paper is to limit the memory available during the computations of such a global optimization code in order to find some efficient feasible solutions. By this way, we introduce a metaheuristic frame to develop some new heuristic global optimization algorithms based on an exact code. We show in this paper, with a small assumption about the sorting by breadth first of elements in the data structure, that the time-complexity of such metaheuristic algorithms becomes polynomial instead of exponential for the exact code. In order to validate our metaheuristic approach, some numerical experiments about constrained global optimization problems coming from the COCONUT library were solved using a heuristic which certifies an enclosure of the global minimum value. The objective is not to solve completely the problem or find a better solution, but it is to know what is the highest precision which can be guaranteed reliably with the available memory.

Keywords Interval Analysis, Branch and Bound, Limited Memory, Metaheuristic, Algorithmic Complexity.

1 Introduction

Nowadays, to solve continuous constrained global optimization problems, we have some different algorithmic strategies depending on: (i) the size of the problem, (ii) some properties about the objective function and of the constraints (for example: linearity, continuity, differentiability, convexity), (iii) the time that we have to solve the problem. These continu-

Jordan Ninin
ENSEEIHT-IRIT, Toulouse, France E-mail: Jordan.Ninin@n7.fr

Frédéric Messine
ENSEEIHT-IRIT, Toulouse, France E-mail: Frederic.Messine@n7.fr

ous constrained global optimization problems can be formulated as follows:

$$\begin{cases} \min\limits_{x \in X \subset \mathbb{R}^n} & f(x) \\ \text{s.t.} & \\ & g_i(x) \leq 0\ , \quad \forall i \in \{1, \ldots, n_g\}, \\ & h_j(x) = 0\ , \quad \forall j \in \{1, \ldots, n_h\}. \end{cases} \tag{1}$$

Assuming that we have enough properties to use interval arithmetic techniques [13], i.e., the equations must be explicitly given, which is a general frame for the study of global optimization problems (all the examples presented in the COCONUT library are so formulated, [2, 15]). Therefore, a large part of deterministic and stochastic, local or global algorithms can be used to try to solve efficiently those kinds of continuous constrained global optimization problems of type (1).

Deterministic global optimization algorithms have already shown their efficiency by solving exactly some of those problems of type (1). Generally the free global optimization software *GlobSol* due to Kearfott et al. is used [8]. *GlobSol* is based on interval arithmetic and Branch and Bound techniques. However, *IBBA* which is our own interval Branch and Bound code have shown its efficiency to solve some industrial problems such as, for example, for the design of electrical machines [5, 6, 10, 12]; in [6] and [12] some improvements of the basic *IBBA* code was done for solving exactly problems of design including a black-box constraint (computed by a finite element code) and also where the size is unknown (depending on the first integer variable). Nevertheless, the exponential time-complexity (and also often in memory) of these deterministic algorithms yields their intrinsic limits. One of the main purpose of this paper, is to offer the possibility to deal more efficiently with those complicated design problems.

Thus, to solve efficiently some global optimization problems, the users must often choose between a local search algorithm (such as for example a Lagrangian increased method), metaheuristic techniques based on local search methods (such as for example Taboo or VNS algorithms) or stochastic global optimization techniques (such as for example genetic algorithms or simulated annealing based techniques).

In this paper, the proposed metaheuristic methodology is based on an exact interval global optimization algorithm and its convergence is enforced by limiting the available memory during its execution. Here, we choose our own *IBBA* code to make our metaheuristic based algorithms. However, our following results could be extended to other bisections or multisections based algorithms such as for interval Branch and Bound codes like GlobSol [8].

The idea to put a limit on the memory seems to be first studied by Casado et al. in [1] showing on a lot of numerical examples of unconstrained global optimization problems the great interest of the use of those kinds of heuristic. In our paper, the methodology is presented as a metaheuristic approach yielding to some distinct but linked algorithms. A particularly interesting heuristic keeping information about the enclosure of the global minimum value is detailed. Moreover, some theoretical results are given about the time-complexity of such algorithms with limited memory and when the data structure is managed by breadth first, this time-complexity is proved to be a polynomial one; the data structure is necessary for storing the current pending leaves of the Branch and Bound tree. The numerical tests, that we are addressed with, concern continuous global optimization problems with constraints in order to illustrate the behavior of an heuristic algorithm which provides an information about the enclosure of the global minimum value at the end of its execution.

In Section 2, the metaheuristic methodology based on the limitation of the memory is presented. Some heuristics are then proposed, and one of them named *Hp*, is entirely detailed; *Hp* has the advantage to enclose the global minimum value found by determining a certified accuracy P which is computed during the execution of the algorithm and which guaranteed the P-global optimality. This section contains also some properties about this heuristic *Hp*. Section 3 is dedicated to the study of the time-complexity of such a metaheuristic methodology. One of the main results of this work concerns theorems about an efficient overestimation of the maximal number of iterations of the main loop of an *IBBA* code, which is a linear function depending on the number of variables and thus, the order of complexity becomes polynomial instead of exponential for *IBBA*. Section 4 validates our metaheuristic methodology approach by testing our algorithm based on the heuristic *Hp* on some numerical examples coming from the COCONUT website [2,15]. These first experimentations are just given to illustrate the behavior of such a metaheuristic algorithm and also to show the potentiality of these approaches to solve larger constrained global optimization problems. Section 5 summarizes the results of the paper.

2 A Metaheuristic for IBBA with Limited Memory

In order to clarify the study of those metaheuristic based algorithms, we first recall in Algorithm 1 the *IBBA* code that we use in this paper, see also [7,8,10,14].

Algorithm 1 stops using two distinct criteria: (i) on the value of the objective function such as $\tilde{f} - \min_{(Z,fz)\in\mathcal{L}} fz \leq \varepsilon_f$ or (ii) on the widths of the boxes remaining in $\mathcal{L}$ such as $\max_{(Z,fz)\in\mathcal{L}} w(Z) \leq \varepsilon_{\mathcal{L}}$, where $w(Z)$ is the width of box Z and is equal to the width of its longest edge. Moreover, we must add another stopping criterion: when the data structure $\mathcal{L}$ becomes empty; this criterion is necessary to stop Algorithm 1 if the problem does not have any feasible solution and it is also useful when we will include our limited-memory algorithm. At line 5 of Algorithm 1, an element of the data structure $\mathcal{L}$ is extracted depending on a heuristic: breadth first, lowest lower bound first, for more details see [4]. If a feasible solution exists, Algorithm 1 could provide only a point $\tilde{z}$ where an upper bound of the global minimum value ($\tilde{f}$) is attained (depending on the time and the memory which are needed to solve the problem); thus sometimes some limits on the total CPU-time and on the available memory can be given by the user and then these stopping criteria can be added.

Our metaheuristic method consists in fixing an upper bound `MaxElts` on the number of elements stored in the data structure $\mathcal{L}$. When this bound is reached, a heuristic is used to limit the size of $\mathcal{L}$. This metaheuristic is completely based on IBBA and, in a first step, it does not change the behavior of the algorithm until the size of $\mathcal{L}$ reaches `MaxElts`. The main idea of the metaheuristic is to allow to *IBBA* to continue the optimization after reaching the available memory. The heuristic is included in the Algorithm 1 mainly in line 21, but it can appear everywhere in the main loop (between lines 4 and 22 of Algorithm 1).

The goal of this idea is to choose which elements are eliminated from $\mathcal{L}$ to reduce its size. Following the metaheuristic principle, many heuristics can be proposed, keeping more or less information about the domain under study. Moreover, unlike classical metaheuristic such as Taboo or VNS, a complete study of the domain is done when the heuristic is used; a lower bound of the objective function over all the boxes and evaluations of constraints are done. Thus, all this information can be used to create a heuristic. For example, some distinct heuristics can be described as follows:

– Insert only one new subbox among the two possible ones in step 12 of the algorithm (for example the subbox which has the lowest lower bound fz),

Algorithm 1 IBBA

1: $X :=$ initial box in which the global minimum is searched, $\{ X \subseteq \mathbb{R}^n \}$
2: $\tilde{f} := +\infty$, denotes the current upper bound for the global minimum,
3: $\mathcal{L} := \{(X, -\infty)\}$, initialization of the data structure of the stored elements, {all elements in $\mathcal{L}$ has two components: Z a box and fz a lower bound of $f(Z)$}
4: **repeat**
5: Extract from $\mathcal{L}$ an element (for example the box with the lowest lower bound or the largest one),
6: Bisect the considered box normal to a direction: V_1, V_2 {Select the component which has the maximal width and bisect it by the middle}
7: **for** $j = 1$ to 2 **do**
8: Pruning of V_j using a **Constraint Propagation Technique** [9],
9: **if** V_j is not empty **then**
10: Compute $fv_j :=$ a lower bound of $f(V_j)$, and all the lower and upper bounds of all the constraints over V_j,
11: **if** $\tilde{f} \geq fv_j$ and no constraint is unsatisfied **then**
12: Insert (V_j, fv_j) in $\mathcal{L}$,
13: $\tilde{f} := \min\{\tilde{f}, f(m)\}$, where m is the middle of V_j, if and only if m satisfies all the constraints,
14: **if** $\tilde{f}$ has changed **then**
15: $\tilde{z} := m$
16: Discard in $\mathcal{L}$ all the couples (Z, fz) such that $fz > \tilde{f}$
17: **end if**
18: **end if**
19: **end if**
20: **end for**
21: {*Possible insertion of Algorithm 2.*}
22: **until** A stopping criterion {choose (i) or (ii) or both} or when $\mathcal{L} = \emptyset$

- Eliminate box with the maximal lower bound fz,
- Eliminate the box with the smallest width,
- Eliminate boxes which get the lowest probability to satisfy constraints, according to a statistical criterion,
- Eliminate boxes which get the lowest probability to contain the global minimum.

A lot of other heuristics can be developed and tested following this metaheuristic methodology.

Remark 1 *Using this metaheuristic methodology, it could appear that deleting elements one by one in the data structure $\mathcal{L}$ involves to considerably increase the CPU-time or not improve at all the speed convergence. Hence, it is preferable to delete a large part of $\mathcal{L}$ when the number of elements of $\mathcal{L}$ reaches* `MaxElts`. *For example, it should be interesting to discard from $\mathcal{L}$, 100 or 1000 elements or 10% of* `MaxElts`.

The heuristic developed in this paper, consists in moving the value of the current solution $\tilde{f}$ in order to keep in $\mathcal{L}$ only elements with lowest lower bounds; we denote this heuristic *Hp* (for Heuristic on the Precision). In order to perform that, a new variable P is inserted. This variable represents the precision obtained on the value of the global minimum and it is modified during the computation. Each time that the size of $\mathcal{L}$ reached the limit `MaxElts`, P is incremented. However, when a new value of $\tilde{f}$ is found, P can decrease. Therefore, at the end of the algorithm, P provides a certified accuracy of the solution $\tilde{f}$, i.e., the global minimum value is guaranteed to be in $[\tilde{f} - P, \tilde{f}]$.

At each iteration of the main loop of Algorithm 1, the following subroutine presented in Algorithm 2 will be executed at line 21 of *IBBA*; this merged algorithm is denoted *IBBA-LM*$_{Hp}$. Between line 2 and line 14 of Algorithm 2, when $\tilde{f}$ has changed and if $\tilde{f} \in$

Algorithm 2 Heuristic on precision: LM_{Hp}

1: (Initialization before the first use of this algorithm: $P := \varepsilon_f$, $\varepsilon_{aux} := \varepsilon_f$ and $i := 1$; `MaxElts`: the limit for the number of elements of $\mathcal{L}$; $\tilde{f}_{prev}$ which is the previous value of the current minimum if $\tilde{f}$ has changed.)
2: **if** $\tilde{f}$ has changed **then**
3: **if** $(\tilde{f} \neq \infty)$ and $(\tilde{f} \geq \tilde{f}_{prev} - P)$ **then**
4: $P := \max\{\tilde{f} - \tilde{f}_{prev} + P, \varepsilon_f\}$
5: $\varepsilon_{aux} := \varepsilon_f$
6: **while** $\varepsilon_{aux} \leq \frac{P}{10 \times i}$ **do**
7: $\varepsilon_{aux} := 10 \times \varepsilon_{aux}$
8: **end while**
9: **else**
10: $P := \varepsilon_f$
11: $\varepsilon_{aux} := \varepsilon_f$
12: $i := 1$
13: **end if**
14: **end if**
15: **if** $(\tilde{f} \neq \infty)$ and $(|\mathcal{L}| \geq$ `MaxElts`**)** **then**
16: **if** $\tilde{f} - P > \min_{(Z,fz)\in\mathcal{L}} fz$ **then**
17: **while** $\tilde{f} - P - \varepsilon_{aux} < \min_{(Z,fz)\in\mathcal{L}} fz$ **do**
18: $\varepsilon_{aux} := \frac{\varepsilon_{aux}}{10}$
19: **end while**
20: **end if**
21: $\varepsilon_{aux} := \max\{\varepsilon_{aux}, \varepsilon_f\}$
22: $P := P + \varepsilon_{aux}$
23: Delete in $\mathcal{L}$ all the elements (Z, fz) such that $fz > \tilde{f} - P$
24: $i := i + 1$
25: **end if**
26: **if** $i = 10$ **then**
27: $\varepsilon_{aux} := 10 \times \varepsilon_{aux}$
28: $i := 1$
29: **end if**

$[\tilde{f}_{prev} - P, \tilde{f}_{prev}]$, then the value of P is updated without modifying the bound $\tilde{f} - P$, i.e., $P := \tilde{f} - \tilde{f}_{prev} + P$; else P is set to the initial value ε_f. Moreover, ε_{aux} is a variable which represents the step to increment P, see line 22. ε_{aux} is updated when the value $\tilde{f}$ changes in line 6-8 or line 11, and it is multiplied by 10 in line 27 to increase P as a logarithmic scale when Hp is used 10 times to not waste too much time in Algorithm 2; in line 17, if all the elements of $\mathcal{L}$ can be eliminated by one step, i.e. $\tilde{f} - (P + \varepsilon_{aux}) \leq \min_{(Z,fz)\in\mathcal{L}} fz$, ε_{aux} is divided by 10 until some elements in $\mathcal{L}$ are kept, i.e. $\tilde{f} - (P + \varepsilon_{aux}) > \min_{(lZ,fz)\in\mathcal{L}} fz$. Thus, during the execution of the algorithm, P can increase if `MaxElts` is reached and decrease if a better value of $\tilde{f}$ is found.

Remark 2 *For IBBA-LM_{Hp}, it is not necessary to delete a great part of $\mathcal{L}$, because it is indirectly provided by increasing the value of P.*

Proposition 3 *At the end of* IBBA-LM_{Hp} *when the stopping criterion* $\tilde{f} - \min_{(Z,fz)\in\mathcal{L}} fz \leq \varepsilon_f$ *is chosen, the global minimum value (if a feasible solution exists) is in* $[\tilde{f} - P, \tilde{f}]$.

Proof. If the number of elements of $\mathcal{L}$ never reaches `MaxElts`, no heuristic included in Algorithm 1 is used and $P = \varepsilon_f$ (the initial value). Hence, Algorithm 1 is used alone and provides certified enclosure of the global minimum value with a width less than $P = \varepsilon_f$.
In the other case, Algorithm 2 is used and rejeted from $\mathcal{L}$ boxes (Z, fz) if $fz > \tilde{f} - P$. At

the beginning of *IBBA-LM*$_{Hp}$, $P = \varepsilon_f$, thus $\tilde{f} - P$ is a certified lower bound of the global minimum value. If P increases (line 22 of Algorithm 2), all elements of $\mathcal{L}$ could be deleted only if it is proved that they cannot provide a solution lower than $\tilde{f} - P$. One denotes P_{prev} the value of P at the previous iteration (P_{prev} corresponds to the value of P before line 4 of Algorithm 2). If $\tilde{f}$ changes, P decreases such as $P := \max\{\tilde{f} - \tilde{f}_{prev} + P_{prev}, \varepsilon_f\}$ (line 4 of Algorithm 2), thus all elements that has been already deleted could not provide a solution lower than $\tilde{f} - P$ because $\tilde{f} - P \leq \tilde{f}_{prev} - P_{prev}$. Indeed, at the end of the algorithm, the deleted elements are the elements which do not satisfied the constraints or cannot provide a solution lower than $\tilde{f} - P$. Therefore, at the end of the algorithm, $\tilde{f} - P$ is a certified lower bound of the global minimum value. Furthermore, $\tilde{f}$ corresponds to a feasible solution, thus it is also an upper bound of the global minimum value.
Finally, the interval $[\tilde{f} - P, \tilde{f}]$ is a certified enclosure of the global minimum value. ■

Remark 4 *In order to use efficiently IBBA-LM*$_{Hp}$*, it could be interesting to search some local minima using a classical local algorithm. Thus, the corresponding value of the best local minima could be used in Algorithm 2 by initializing $\tilde{f}$ to this corresponding value. When no feasible solution will be provided, it is possible that the number of elements in $\mathcal{L}$ during the running of Algorithm 1 including Algorithm 2, can be greater than the limit* `MaxElts` *and so on, until a first feasible solution is obtained. In this case, this heuristic is inefficient and another one should be added.*

3 Study of the Complexity in Time for some *IBBA-LM* Algorithms

In this section, we study the time-complexity and memory-complexity of such metaheuristic algorithms denoted by *IBBA-LM* (Algorithm 1 and 2). Thus, we show that when the data structure $\mathcal{L}$ of boxes under study is managed by breadth first, these metaheuristic based algorithms with a limitation on memory involves a change of the time-complexity, i.e., from exponential, it becomes a linear one by considering uniquely the number of iterations of the main loop of Algorithm 1 (from step 4 to 22).

In the following, we denote by *IBBA*$^{\texttt{bf}}$ and *IBBA-LM*$^{\texttt{bf}}$ the algorithms where the data structure is managed by breadth first, i.e., where the first element is the one with the largest width $w(Z)$ (this order is chosen because it is independent of the computation of the lower bounds fz). The notations $IBBA_{\varepsilon_f}(X)$ and $IBBA_{\varepsilon_{\mathcal{L}}}(X)$ are also used in this paper to indicate that an *IBBA* algorithm works with an initial search domain which is a box X and uses a stopping criterion respectively: (i) on the accuracy on the value of the minimum, denoted by ε_f; (ii) on the width of the elements of $\mathcal{L}$, denoted by $\varepsilon_{\mathcal{L}}$.

Let call $P_{IBBA}(n)$ the complexity inside the main loop of an *IBBA* algorithm for an instance of problem (1) with n variables. This complexity $P_{IBBA}(n)$ depends on what procedures are used in the *IBBA* code, i.e., if the constraint propagation technique is used or not [9], what kind of techniques are used for computing bounds [11,14], etc. However, this complexity is polynomial in time and the corresponding subroutines use a small part of the memory for those computations, see [9,11] for some examples of such accelerating subroutines including studies of complexity.

In this section, we use the classical definition in interval analysis of the width of a box Z: $w(Z) = \max_{i\in\{1,\ldots,n\}} w(Z_i) = \max_{i\in\{1,\ldots,n\}} (\overline{z_i} - \underline{z_i})$, where $Z_i = [\underline{z_i}, \overline{z_i}]$ is the ith component of the box Z, and where $w(Z_i) = (\overline{z_i} - \underline{z_i})$ is the width of the i^{th} component Z_i of the box Z.

First, we are interested in a particular property on the full binary tree, denoting by $\mathcal{T}(X)$, which is created from the recursive bisections by the middle of the largest component of the

boxes. The binary tree $\mathcal{T}(X)$ is constructed until the width of all the so-generated subboxes becomes less than $\varepsilon_{\mathcal{L}}$. In the following, we use the classical definition of a level of a binary tree; in level 0, we have only X, in level 1 we have just the two subboxes generated by a bisection of X by the middle of its largest component, etc. At each level k, we have 2^k subboxes where each of them is generated by k successive bisections of X.

Proposition 5 *The width of all components of all subboxes in a same level of $\mathcal{T}(X)$ are equal.*

The idea of this proof is to demonstrate that if two subboxes on the same level of $\mathcal{T}(X)$ differ by the width of at least one component, then its predecessors also differ by the width of at least one component and so on, until the root which must be different and thus this provides a contradiction.

Proof. First we note that two subboxes issued from the bisection of a same box have exactly the same components except one which has the same width (bisection by the middle of this component); these two subboxes are in the same level of $\mathcal{T}(X)$.

Now suppose that there exist two subboxes Z^1 and Z^2 which are both on the same level k of $\mathcal{T}(X)$. Z^1 and Z^2 differ by the width of at least one component; let denote the corresponding direction by ν. Therefore, this implies, following the first sentence of this proof, that Z^1 and Z^2 comes from the decomposition of two distinct boxes in level $k-1$ of $\mathcal{T}(X)$; let denote these two predecessors by W^1 and by W^2 respectively. At least one of the widths of all the components of W^1 must differ from those of W^2 by only four distinct ways: (i) W^1 and W^2 have the same ν^{th} component of Z^1 and respectively Z^2, hence W^1 and W^2 differ by width in this direction; (ii) W^1 has the same component of Z^1 in direction ν (where the width differs from Z^2) but not W^2, i.e., the width of the component in direction ν of Z^2 is multiplied by 2 in W^2. This implies that in W^1 another direction $\mu \neq \nu$ was chosen for performing the bisection and therefore the widths of the components in direction μ in W^1 and W^2 differ; (iii) W^2 has the same component in direction ν but not W^1, by a similar reasoning from (ii), we conclude that there exist two components with different widths in W^1 and W^2; (iv) W^1 and W^2 do not have the same ν^{th} component of respectively Z^1 and Z^2, that implies that in both cases the bisection was done by this same direction ν and these two components in W^1 and W^2 have twice the widths of those in Z^1 and respectively Z^2, and thus they also differ by width.

Hence by recursion, we arrive to the two first subboxes coming from the bisection of the box X which must have exactly the same widths for all the components (following the first sentence of this proof). This is a contradiction and the result follows. ∎

Corollary 6 *All subboxes of a same level in $\mathcal{T}(X)$ have exactly the same width.*

Proof. This corollary is just a consequence of Proposition 5. ∎

3.1 Stopping criterion on the width

In this subsection, we take into account only the stopping criterion on the widths of boxes:

$$\max_{(Z,fz)\in\mathcal{L}} w(Z) \leq \varepsilon_{\mathcal{L}}, \tag{2}$$

where $\varepsilon_{\mathcal{L}}$ is a small positive value which is given by the user of Algorithm 1. We denote $IBBA_{\varepsilon_{\mathcal{L}}}$ the algorithm using this stopping criterion, see equation (2).

At each iteration of the main loop of $IBBA_{\varepsilon_\mathcal{L}}$ with or without the limitation of the memory, the element is bisected into two subboxes. Thus we have 3 cases: (i) the 2 subboxes are inserted in $\mathcal{L}$, (ii) only one subbox is inserted in $\mathcal{L}$, (iii) no subbox is inserted. The initial box which provides these two subboxes are discarded from $\mathcal{L}$ and therefore, in the worst case (without elimination and reduction of subboxes), the number of elements in $\mathcal{L}$ increases of one per iteration of the main loop of $IBBA_{\varepsilon_\mathcal{L}}$. Hence, in the worst case and without the use of the limitation on memory technique, the number of elements in $\mathcal{L}$ is equal to the number of iteration of the main loop of $IBBA_{\varepsilon_\mathcal{L}}$ plus one.

Proposition 7 *By denoting $\mathcal{N}(\text{IBBA}_{\varepsilon_\mathcal{L}}(X))$ the maximal number of iteration of an* $\text{IBBA}_{\varepsilon_\mathcal{L}}$ *code used for solving a problem with an initial domain denoted by X, we have:*

$$\mathcal{N}(\text{IBBA}_{\varepsilon_\mathcal{L}}(X)) = 2^{\sum_{i=1}^{n}\left\lceil \log_2 \frac{w(X_i)}{\varepsilon_\mathcal{L}}\right\rceil} - 1, \tag{3}$$

and an efficient overestimation is:

$$\mathcal{N}(\text{IBBA}_{\varepsilon_\mathcal{L}}(X)) \leq 2^{n\times\left\lceil \log_2 \frac{w(X)}{\varepsilon_\mathcal{L}}\right\rceil}. \tag{4}$$

Proof. By remarking that the maximal number of iterations of the main loop of an $IBBA_{\varepsilon_\mathcal{L}}$ code which is used to solve a problem with an initial domain X is obtained when all the search tree $\mathcal{T}(X)$ is constructed, then it is just necessary to count the number of iterations to construct completely $\mathcal{T}(X)$. To have all the subboxes Z with a width $w(Z)$ less than $\varepsilon_\mathcal{L}$, it is necessary to bisect K_i times each component i until $\frac{w(Z_i)}{2^{K_i}} \leq \varepsilon_\mathcal{L}$ yielding $K_i := \left\lceil \log_2 \frac{w(Z_i)}{\varepsilon_\mathcal{L}}\right\rceil$. Starting with all the decomposition of each component, we generate $\prod_{i=1}^{n} 2^{K_i}$ subboxes with a size less than $\varepsilon_\mathcal{L}$. Using Corollary 6, one has that at each level of $\mathcal{T}(X)$ all the boxes have the same width, hence at the last level of $\mathcal{T}(X)$ we must have these $\prod_{i=1}^{n} 2^{K_i} = 2^{\sum_{i=1}^{n} K_i}$ subboxes with a width less than $\varepsilon_\mathcal{L}$. By denoting the level 0, the first level in $\mathcal{T}(X)$ which just contains X, we obtain a link between the number of the level and the number of elements in its level; i.e., for level k we have 2^k elements at this level k of $\mathcal{T}(X)$. Therefore, the last level of $\mathcal{T}(X)$ is the $\sum_{i=1}^{n} K_i$ one. To count the number of bisections in $\mathcal{T}(X)$ which corresponds to the maximal number of iterations of the main loop in an $IBBA_{\varepsilon_\mathcal{L}}$ code, we have to sum all the elements of each level of $\mathcal{T}(X)$ until the last level minus one:

$$\mathcal{N}(IBBA_{\varepsilon_\mathcal{L}}(X)) = \sum_{l=0}^{\sum_{i=1}^{n} K_i - 1} 2^l = 2^{\sum_{i=1}^{n} K_i} - 1,$$

and the first result follows. For the overestimation, it directly occurs when the inequality $w(X_i) \leq w(X), \forall i \in \{1, \cdots, n\}$ is used. ∎

In order to simplify, we can consider the order of complexity of $IBBA_{\varepsilon_\mathcal{L}}$ which is given by the following proposition.

Corollary 8 *The order of time-complexity of an* $\text{IBBA}_{\varepsilon_\mathcal{L}}$ *algorithm is:*

$$\mathcal{O}(\text{IBBA}_{\varepsilon_\mathcal{L}}(X)) = 2^n. \tag{5}$$

To be more precise, we have to multiply this complexity by $P_{IBBA_{\varepsilon_\mathcal{L}}}(n)$ but the order is 2^n and this confirms the exponential complexity of an IBBA *algorithm.*

Proof. This result is directly obtained from equation (4) of Proposition 7. ∎

Remark 9 *Note that this order of complexity is the same even if different heuristics for the management of the data structure $\mathcal{L}$ are used: by deeper or breadth first, or by sorting $\mathcal{L}$ by the increasing sort of the lower bounds fz, etc. Of course, as interval arithmetic has a contraction mapping property [14], generally more boxes will be deleted using a heuristic of deeper first in place of breadth first.*

The previous theoretical results are well-known and some of them can be found in [3]; here, they are recalled and adapted to our notations and an *IBBA*$_{\varepsilon_{\mathcal{L}}}$ code. The following properties are new and deal with the complexity induced by the use of the metaheuristic algorithmic frame proposed in this paper.

Now, we study the complexity of our metaheuristic based algorithm denoted by *IBBA-LM*. The following results of time-complexity are only obtained for *IBBA-LM*$^{\texttt{bf}}$ (where the data structure is managed by breadth first), which is a particular but a general case of *IBBA-LM* algorithms. Let denote by $\mathcal{N}(IBBA(X))$ the maximal number of iterations of the main loop of a general interval branch and bound algorithm *IBBA* including or not a limited memory metaheuristic technique.

Proposition 10 *If* $\mathcal{N}(\text{IBBA}_{\varepsilon_{\mathcal{L}}}(X)) \leq$ `MaxElts` *then*

$$\mathcal{N}(\text{IBBA-LM}_{\varepsilon_{\mathcal{L}}}(X)) = \mathcal{N}(\text{IBBA}_{\varepsilon_{\mathcal{L}}}(X)) = 2^{\sum_{i=1}^{n}\left\lceil \log_2 \frac{w(X_i)}{\varepsilon_{\mathcal{L}}}\right\rceil} - 1,$$

i.e., if the maximal number of iterations of IBBA$_{\varepsilon_{\mathcal{L}}}$ *is less than* `MaxElts`, IBBA$_{\varepsilon_{\mathcal{L}}}$ *and* IBBA-LM$_{\varepsilon_{\mathcal{L}}}$ *has the same maximal number of iterations.*

Proof. It is evident because the limit on the memory is never reached and therefore the heuristic of *IBBA-LM*$_{\varepsilon_{\mathcal{L}}}$ is never used and the iterations of *IBBA-LM*$_{\varepsilon_{\mathcal{L}}}$ and of *IBBA*$_{\varepsilon_{\mathcal{L}}}$ are exactly the same (of course if the main iterations inside *IBBA-LM*$_{\varepsilon_{\mathcal{L}}}$ and *IBBA*$_{\varepsilon_{\mathcal{L}}}$ are exactly the same). Thus, in the worst case (generating the maximal number of iterations), this result follows. ∎

Hence, if `MaxElts` is too high our metaheuristic methodology will never be used. However, this metaheuristic will be used in order to never reach the capacity in memory of the computer.

Let now consider the interesting case when the capacity in memory of the computer is reached using an *IBBA* code and thus, when the limitation of the memory used involves some changes on the behavior of *IBBA*.

Theorem 11 *An efficient overestimation value for the maximal number of iterations of the main loop of such an* IBBA-LM$^{\texttt{bf}}_{\varepsilon_{\mathcal{L}}}$ *metaheuristic algorithm is given by:*

$$\mathcal{N}(\text{IBBA-LM}^{\texttt{bf}}_{\varepsilon_{\mathcal{L}}}(X)) \leq \texttt{MaxElts} \times \left(\sum_{i=1}^{n}\left\lceil \log_2 \frac{w(X_i)}{\varepsilon_{\mathcal{L}}}\right\rceil\right). \tag{6}$$

Proof. The maximum number of evaluated boxes is bounded above by the maximum number of boxes per level of $\mathcal{T}$ which are allowed to be stored in $\mathcal{L}$ (`MaxElts`) times the maximum number of levels of $\mathcal{T}$. ∎

In fact, to be more precise, we have that :
$\mathcal{N}(IBBA\text{-}LM^{\texttt{bf}}_{\varepsilon_{\mathcal{L}}}(X)) \leq \min\left\{\mathcal{N}(IBBA_{\varepsilon_{\mathcal{L}}}(X)), \texttt{MaxElts} \times \left(\sum_{i=1}^{n}\left\lceil \log_2 \frac{w(X_i)}{\varepsilon_{\mathcal{L}}}\right\rceil\right)\right\}$.
However, here we just study the time-complexity when the limitation of the memory has an effect in such an *IBBA* code.

Theorem 12 *The order of time-complexity of such* IBBA-LM$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$ *metaheuristic algorithm is:*

$$\mathcal{O}(\text{IBBA-LM}^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}(X)) = n \times P_{\textit{IBBA-LM}_{\varepsilon_{\mathcal{L}}}}(n). \tag{7}$$

To be more accurate this order of complexity must be multiplied by a constant which can have an important value: $\mathtt{MaxElts} \times \left\lceil \log_2 \frac{w(X)}{\varepsilon_{\mathcal{L}}} \right\rceil$.

Proof. Using Theorem 11, we can make another overestimation by considering the fact that $\forall i \in \{1,\cdots,n\}, w(X_i) \leq w(X)$. Thus

$$\mathtt{MaxElts} \times \left(\sum_{i=1}^{n} \left\lceil \log_2 \frac{w(X_i)}{\varepsilon_{\mathcal{L}}} \right\rceil \right) \leq n \times \mathtt{MaxElts} \times \left\lceil \log_2 \frac{w(X)}{\varepsilon_{\mathcal{L}}} \right\rceil,$$

and the result follows by multiplying by $P_{\textit{IBBA}-\textit{LM}}(n)$ the polynomial complexity inside the main loop of *IBBA-LM*$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$. ∎

Consider $\mathcal{A}_{\max}$ an *IBBA-LM*$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$ algorithm which generates the maximal number of iterations. Denote $\mathcal{L}$ a data structure of stored boxes under study managed by breadth first and $\mathcal{L}(k)$ the data structure obtained after k iterations. If using less than k iterations the solution is obtained then we take by convention $\mathcal{L}(k) = \emptyset$ and $\max_{(Z,fz)\in\mathcal{L}(k)} w(Z) = 0$. $\mathcal{A}_{\max}$ uses a data structure denoted by $\mathcal{L}_{\max}$. At each iteration of the main loop, no elimination and no reduction of a subbox is done (for an *IBBA-LM*$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$ algorithm the decomposition is performed by breadth first and the bisections by the middle of the largest component of the largest subbox). Hence, the size of $\mathcal{L}_{\max}$ will rapidly increase until the limit `MaxElts` is reached. We will show below that this algorithmic behavior will generate the largest number of iterations of the main loop for an *IBBA-LM*$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$ method. This will be used to study the complexity of such an algorithm.

Lemma 13 *Let denote $\mathcal{L}$ a data structure generated during the run of an* IBBA-LM$^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}$. *We have:*

$$\max_{(Z,fz)\in\mathcal{L}(k)} w(Z) \leq \max_{(Z,fz)\in\mathcal{L}_{\max}(k)} w(Z), \forall k \in \{1,\cdots,\mathcal{N}(\text{IBBA}_{\varepsilon_{\mathcal{L}}}(X))\}.$$

Proof. First we note that at iteration $\mathtt{MaxElts}-1$, one has `MaxElts` elements in $\mathcal{L}_{\max}$ which is the maximum (all the subboxes are kept) and therefore, from iteration 1 to iteration $\mathtt{MaxElts}-1$, one has $\max_{(Z,fz)\in\mathcal{L}(k)} w(Z) \leq \max_{(Z,fz)\in\mathcal{L}_{\max}(k)} w(Z)$.

In the following, $l_{\inf}(\mathcal{A},k)$ will denote the minimum explored level in $\mathcal{T}(X)$ for the algorithm $\mathcal{A}$ at iteration k (if we are in some different levels, the level minimum is returned). Furthermore, we denote by $n_{\max}$ the number of elements in level l of $\mathcal{T}(X)$ stored in $\mathcal{L}_{\max}$ and by $l = l_{\inf}(\mathcal{A}_{\max}, \mathtt{MaxElts}-1)$.

At iteration $\mathtt{MaxElts}-1+n_{\max}$, one has in the worst case, `MaxElts` elements in $\mathcal{L}_{\max}$; the limit is reached since iteration $\mathtt{MaxElts}-1$. Thus, one has $l_{\inf}(\mathcal{A}_{\max}, \mathtt{MaxElts}-1+n_{\max}) = l+1$. Indeed, between the iteration $\mathtt{MaxElts}-1$ and iteration $\mathtt{MaxElts}-1+n_{\max}$, the $n_{\max}$ elements extracted from $\mathcal{L}_{\max}$ are the $n_{\max}$ elements of the level l of $\mathcal{T}(X)$ ($\mathcal{L}$ is sorted by breadth first), and to not reach the limit `MaxElts` and to not use the heuristic, we can only add one of two subboxes generated by the bisection.

Moreover, one has $l_{\inf}(\mathcal{A}, \mathtt{MaxElts}-1+n_{\max}) \geq l+1$ (where $\mathcal{A}$ is the algorithm corresponding to $\mathcal{L}$), because if $l_{\inf}(\mathcal{A}, \mathtt{MaxElts}-1) = l$, this implies that the number of elements in $\mathcal{L}$, at this level l and at iteration $\mathtt{MaxElts}-1$, is less than $n_{\max}$ which is impossible following the construction of $\mathcal{L}_{\max}$ where each generated subboxes is kept until iteration $\mathtt{MaxElts}-1$. After $k \times \mathtt{MaxElts}$ more iterations, one has $l_{\inf}(\mathcal{A}_{\max}, \mathtt{MaxElts}-1+n_{\max}+$

$k \times \texttt{MaxElts}) = l + 1 + k$. One notes that it is impossible to go down through $\mathcal{T}(X)$ slowly (in term of increasing the level) without the need of strictly more than `MaxElts` elements by level of $\mathcal{T}(X)$ which is impossible by the use of our metaheuristic methodology based on the limitation of the number of elements in the data structure at `MaxElts`.

In conclusion, it is impossible to have at an iteration $k > \texttt{MaxElts} - 1$, $l_{\inf}(\mathcal{A}_{\max}, k) > l_{\inf}(\mathcal{A}, k)$ which implies, following Proposition 5, to have a case where it exists a subbox $W \in \mathcal{L}(k)$ such that $w(W) > \max_{(Z,fz)\in\mathcal{L}_{\max}(k)} w(Z)$. Hence, the result follows. ∎

Remark 14 *These results are always true for all possible* IBBA-LM$^{\texttt{bf}}_{\varepsilon_{\mathcal{L}}}$ *algorithms including or not accelerating techniques, such as constraint propagation techniques. This is due to the fact that Algorithm* $\mathcal{A}_{\max}$ *will always generates the maximum number of iterations comparing to all possible kinds of* IBBA-LM$^{\texttt{bf}}_{\varepsilon_{\mathcal{L}}}$ *algorithms. Furthermore, these properties are still true if we consider interval Branch and Bound algorithms for solving unconstrained global optimization problems such as those described in [1]; it is just necessary to have a bisection technique by the middle of the largest component of a box, and the data structure* $\mathcal{L}$ *must be sorted by breadth first. Thus, those theoretical results partially explain why the fast interval algorithms presented in [1] are so efficient.*

At this point, we cannot say anything if the management of the data structure $\mathcal{L}$ is done by another way than breadth first, such as for example by deeper first or by lowest lower bound first.

3.2 Stopping criterion on the precision of the optimal value

Generally in *IBBA* we use the following test to stop the run of a code when we want to precisely enclose the optimal value:

$$\tilde{f} - \min_{(Z,fz)\in\mathcal{L}} fz \leq \varepsilon_f, \tag{8}$$

where $\tilde{f}$ is a current feasible solution in *IBBA* (it is found during the computations, see Algorithm 1).

In oder to obtain the following results about the time-complexity, we have to assume that the global minimum value, denoted by f^*, is found, i.e., $f^* = \tilde{f}$. Using the definition of the $\alpha-$convergence [8, 14, 16] of the inclusion functions used for computing bounds of f over a box Z, and by denoting fz the corresponding lower bound which is generally also stored in $\mathcal{L}$, one has the following inequality:

$$\forall (Z, fz) \in \mathcal{L},\ f^* - fz \leq w(F(Z)) \leq a \times w(Z)^{\alpha}, \tag{9}$$

where $F(Z)$ is an inclusion function of the function f over the box Z providing an interval which encloses $f(Z)$ (the range of f over Z) and where a and α are two strictly positive real constant values.

If $\overline{F(Z)} \leq f^*$, the box Z could be deleted because it does not verify a constraint since f^* is the global minimum value; and if $\underline{F(Z)} > f^*$, the elimination step of line 16 of Algorithm 1 should eliminate the box Z from $\mathcal{L}$. Indeed, $\forall (Z, fz) \in \mathcal{L},\ f^* - fz \leq w(F(Z))$. Therefore, in the worst case, if we want to major $\left(f^* - \min_{(Z,fz)\in\mathcal{L}} fz\right)$ by ε_f, we can overestimate $a \times w(Z)^{\alpha}$ by ε_f. Hence, one obtains that the criterion formulated by equation (8) can be written as:

$$w(Z) \leq \left(\frac{\varepsilon_f}{a}\right)^{\frac{1}{\alpha}} \tag{10}$$

Theorem 15 *With a stopping criteria on the accuracy of the optimal value, an efficient overestimation of the maximal number of iterations of the main loop of* IBBA-LM$^{\mathtt{bf}}$ *denoting in this case* IBBA-LM$^{\mathtt{bf}}_{\varepsilon_f}$ *is:*

$$\mathcal{N}\left(\text{IBBA-LM}^{\mathtt{bf}}_{\varepsilon_f}(X)\right) = \mathtt{MaxElts} \times \left(\sum_{i=1}^{n} \left\lceil \log_2 \frac{w(X_i)}{\left(\frac{\varepsilon_f}{a}\right)^{\frac{1}{\alpha}}} \right\rceil \right) \tag{11}$$

and the corresponding order of time-complexity is:

$$\mathcal{O}\left(\text{IBBA-LM}^{\mathtt{bf}}_{\varepsilon_f}(X)\right) = n \times P_{IBBA\text{-}LM^{\mathtt{bf}}_{\varepsilon_{\mathcal{L}}}}(n). \tag{12}$$

To be more accurate this order of complexity must be multiplied by a constant which can have an important value: $\mathtt{MaxElts} \times \left\lceil \log_2 \frac{w(X)}{\left(\frac{\varepsilon_f}{a}\right)^{\frac{1}{\alpha}}} \right\rceil$.

Proof. We obtain this proposition directly from equation (10) and by applying Theorems 11 and 12. ∎

4 Numerical Experiments

In this section, we study our algorithm denoted by *IBBA-LM$_{Hp}$* . This one combines Algorithm 1 and 2 based on the metaheuristic principle described in Section 2. Its data structure $\mathcal{L}$ is sorted by the increasing order of the lower bounds (fz) and its stopping criterion is on the enclosure of the value of the global minimum, see equation (8). This section is divided into two subsections. The first one deals with an example due to H. Tuy and presents some detailed discussions about the behavior of algorithm *IBBA-LM$_{Hp}$* using different values for the limitation of the memory. In the second subsection, we test the efficiency of our *IBBA-LM$_{Hp}$* code on some test problems issued from the Library 1 of the COCONUT website [2, 15], which have from 11 to 33 variables.

All the tests are performed on a PC-Intel-Xeon-3GHz computer with 2GB of RAM and using a 64-bit Linux system. The code is made in Fortran 90/95 using f90 which is a SUN compiler including the interval arithmetic library.

4.1 An example of Tuy

The following example is due to H. Tuy; it can be also found in the Library 2 of the COCONUT website [2], denoted by *hs071*:

$$\begin{cases} \min\limits_{x\in[1;5]^4} & x_3 + (x_1 + x_2 + x_3)x_1x_4 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ & x_1x_2x_3x_4 \geq 25 \end{cases} \tag{13}$$

In Table 1, we solve this problem using Algorithm 1 with a required precision of $\varepsilon_f = 10^{-8}$ on the value of the minimum and of the constraints. We compare the results obtained by *IBBA-LM$_{Hp}$* using a limit on the maximum number of elements in $\mathcal{L}$ about 250 000, 500 000 and 750 000 which correspond to some approximative values in MegaBytes indicated in

Table 1. *Nits* corresponds to the number of iterations of the main loop of Algorithm 1. The precision P represents the accuracy which is certified at the end of *IBBA-LM*$_{Hp}$; e.g., the global minimum value is guaranteed to be between the solution minus P and the solution.

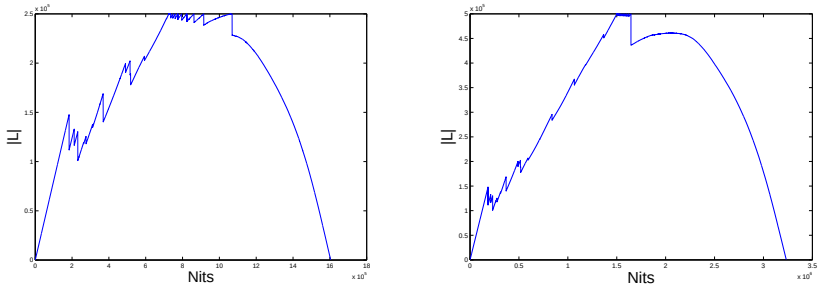
In Table 1, we remark that even if the limit on the memory is very low `MaxElts` $=$ 250 000, *IBBA-LM*$_{Hp}$ solved the problem with a very efficient accuracy which is less than 10^{-5} and with less than 2 minutes. Thus, it is shown on this example that a small certified enclosure of the global minimum value can be obtained without the need to use a large part of the memory of the computer. On this example, *IBBA* needs 2GB of memory which is exactly the physical limit of our computer, see Table 1. Therefore, the operating system can swap some elements of $\mathcal{L}$ on the hard disk and the efficiency in CPU-time can be strongly degraded; there is a factor 1 000 between an access in memory compared to an access on a hard disk. This is maybe the case on this example: 47 minutes is needed to solve the problem, see Table 1.

Algo. `MaxElts`	*IBBA-LM*$_{Hp}$ 250 000	*IBBA-LM*$_{Hp}$ 500 000	*IBBA-LM*$_{Hp}$ 750 000	*IBBA* (no limit)
max $\lvert\mathcal{L}\rvert$ RAM	250 000 128MB	500 000 256MB	750 000 512MB	$>4\,000\,000$ 2GB
Nits	1 698 543	3 232 935	4 968 019	22 710 768
CPU Time	1.72*min*	3.63*min*	7.28*min*	46.96*min*
Solution	17.0140175910	17.0140173514	17.014017320914	17.0140172872
P	6.0089×10^{-6}	1.652×10^{-6}	7.05676×10^{-7}	1×10^{-8}

Table 1 Behavior of *IBBA-LM*$_{Hp}$ on a Tuy's example

In Figure 1, we represent the behavior of *IBBA-LM*$_{Hp}$ on the Tuy's example (13) by limiting the number of elements in $\mathcal{L}$ with 250 000 and 500 000 elements; the number of stored elements $\lvert\mathcal{L}\rvert$ over the number of iterations *Nits* are drawn. In a first step, we note that the size of $\mathcal{L}$ increases following an exponential complexity. Some asperities appear in Figure 1 when a better feasible solution is found, which makes it possible to delete some elements from $\mathcal{L}$. When the limit `MaxElts` is reached during a second step, *IBBA-LM*$_{Hp}$ remains very close to this limit. The variable P of Algorithm 2 is then increased many times and then in the third step, the algorithm converges by eliminating with a lower bound over $\tilde{f}-P$. This point shows that the convergence of an *IBBA* code is then enforced.

Fig. 1 Behavior of the size of $\mathcal{L}$ over the number of iterations with `MaxElts` $=$ 250 000 and `MaxElts` $=$ 500 000.



4.2 Some numerical tests from Library 1 of COCONUT

In order to illustrate the behavior of our algorithm *IBBA-LM*$_{Hp}$, we have selected some examples from Library 1 of COCONUT [2]. The number of variables is between 11 and 33, which is quite large for such *IBBA* algorithms. To focus our study on the heuristic *Hp*, we give to *IBBA-LM*$_{Hp}$ the best known upper bound of the global solution given in [2] in place of $+\infty$, i.e., at the beginning of *IBBA-LM*$_{Hp}$, $\tilde{f}$ corresponds to the best feasible solution which is previously found by solvers such as BARON, MINOS, DONLP2 or LINGOS, see [2]. The values of these non-guaranteed upper bounds are reported in column *BestSol* of Table 2. This approach could be easily generalized: the best known solution should be obtained by another algorithm (for example coming from a Taboo or a VNS code) and *IBBA* is then used to prove that it is the best solution and moreover *IBBA* can also improve it. Thus, the objective is not to solve completely the problem or find a better solution, but it is to know what is the highest precision which can be guaranteed reliably with the available memory.

In Table 2, all the tests are presented. N denotes the number of variables and M the number of constraints. P represents the certified accuracy that is obtained at the end of the run of an *IBBA* code (defined on the top of the column), that means when the algorithm finishes the global minimum is certified reliably in $[BestSol - P, BestSol]$ (for all the examples *BestSol* was not improved). We perform several tests on each examples with different values for `MaxElts`, and the last one without this heuristic *Hp*. We choose $\tilde{f} - \min_{(Z,fz)\in\mathcal{L}} fz \leq \varepsilon_f$ as stopping criterion. We start with $\varepsilon_f = 10^{-6}$ except for Tuy's example where $\varepsilon_f = 10^{-8}$. We add a stopping criterion on the CPU-time fixed to 60 minutes. For only *IBBA* without *Hp* code, the algorithm stops if the size of $\mathcal{L}$ reaches 2 000 000 elements. If the algorithm stops due to one of this two previous stopping criteria, the value of P displayed in Table 2 is $\max\{P,\ \tilde{f} - \min_{(Z,fz)\in\mathcal{L}} fz\}$.

In Table 2, we note clearly that the accuracy P is better when the value `MaxElts` increases, except on few tests due to the stopping criterion on the CPU-time. For all the examples, *IBBA* and *IBBA-LM*$_{Hp}$ have exactly the same behavior until the size of $\mathcal{L}$ reaches `MaxElts`, see Proposition 10. However, when `MaxElts` $=$ 2 000 000, *IBBA* without using *Hp* must stop because there is no more available memory on the computer and *IBBA-LM*$_{Hp}$ continues yielding much efficient certified accuracies P. We show that with this heuristic it is possible to have more information at the end of the execution of such a code than without its use. Moreover, to obtain the same information about the certified accuracy P, less memory is required than with the direct use of *IBBA* code. Actually, the accuracy obtained with *IBBA-LM*$_{Hp}$ with `MaxElts` $=$ 1 500 000 is better than the accuracy obtained with *IBBA* only. We also note that for all these examples that *IBBA* reaches rapidly the fixed limit of memory (2 000 000 of elements in $\mathcal{L}$): that needs generally a few minutes (less than 15).

In this first study, the used *IBBA* code is very simple, see [14]: (i) just a bisection by the middle of the largest component of a box is used; (ii) $\mathcal{L}$ is sorted by lowest lower bounds first; (iii) a constraint propagation technique detailed in [9] is included; (iv) the bounds are computed directly using the natural extension into intervals of expressions of the function and constraints given in [2]. Thus to make some efficient algorithms to solve the problems presented in Table 2 and also for other examples, we must add some steps of a local search algorithm to improve the current minimum $\tilde{f}$ at some iterations of the *IBBA* code, we also have to test all the heuristics that we proposed in Section 2 and we have to use better methods for computing lower bounds [11, 14, 16].

Table 2 Behavior of *IBBA-LM*$_{Hp}$ on Tuy's problem and on several examples from Library 1 of COCONUT

			MaxElts	*IBBA-LM*$_{Hp}$ 500 000		*IBBA-LM*$_{Hp}$ 1 000 000		*IBBA-LM*$_{Hp}$ 1 500 000		*IBBA-LM*$_{Hp}$ 2 000 000		*IBBA* without *Hp* 2 000 000	
name	N	M	*BestSol* of [2]	*P*	T(min)	*P*	T(min)	*P*	T(min)	*P*	T(min)	*P*	T(min)
TUY	4	2	17.01401728	1.6520e-6	3.63	3.59487e-7	7.57	1.70085575e-7	10.71	8.62882376e-8	32.26	4.45850297e-7	6.03
ex2_1_3	13	9	-15.0	2.0	2.48	0.9	11	0.7	30.33	0.6	47.08	1.3984375	1.75
ex2_1_7	20	10	-4150.4101	610.0	22.47	500.0	42.8	430.957	60	457.7148	60	673.814200831	16.68
ex2_1_8	24	10	15639.0	11000.0	6.97	9000.0	39.55	9047.25	60	8280.4375	60	10283.4375	3.61
ex5_2_5	32	19	-3500	12100.0	15.03	12000.0	21.18	12000.0	24.03	11450	60	11950	8.81
ex6_1_3	13	9	-0.3525	0.5	2.48	0.3	7.95	0.3	25.28	0.3	13.13	0.37225180	2.93
ex8_4_5	15	11	3.e-4	3.e-5	3.76	2.e-5	12.05	2.e-5	25.28	2.e-5	32.73	2.56827015e-5	3.5
ex9_1_4	11	9	-37	4.97951e-5	1.25	3.e-5	2.93	2.e-5	8.33	2.e-5	9.88	2.03741893e-5	1.75
harker	20	7	-986.51350	900.0	7.22	800.0	33.46	800.0	44.55	800.0	31.91	905.61489392	3.71
hydro	31	24	4366944.0	200000.0	5.72	160000.0	19.38	140000.0	43.55	130303.5687	60	158596.7221	5.25
ramsey	33	22	-2.48750	0.07	11.22	0.07	11.33	0.06083178	60	0.06234392	60	0.06707264	7.06

5 Conclusion

In this paper, we present a metaheuristic methodology based on the limitation of the memory used by an *IBBA* algorithm in order to solve efficiently constrained global optimization problems. A theoretical study about the time-complexity of such metaheuristic algorithms when the data structure is managed by breadth first is done yielding to one of the main results: the order of complexity is $n \times P_{IBBA}(n)$ (n times the complexity inside the main loop of an *IBBA* code). This is a polynomial complexity compared to the exponential one of *IBBA*. The metaheuristic approach is validated by considering one heuristic which is entirely detailed and which has the advantage to certify an accuracy provided at the end of the execution of such an algorithm, which encloses the global minimum value. This heuristic is just based on a simple *IBBA* code using directly interval arithmetic for computing bounds and only a code of interval constraint propagation to accelerate its convergence. The purpose of this first study was just to present the general metaheuristic frame, a complete theoretical study of the time-complexity of such algorithms and some interests about the use of these techniques to find efficient solutions of global optimization problems.

References

1. L.G. Casado, J.A. Martínez, I. García, *Experiments with a new selection criterion in a fast interval optimization algorithm*, Journal of Global Optimization, Vol. 19, pp. 247–264, 2001.
2. COntinuous CONstraints - Updating the Technology, COCONUT website: http://www.mat.univie.ac.at/users/neum/public_html/glopt/coconut/
3. A.E. Csallner, T. Csendes and M.C. Markot, *Multisection in interval branch-and-bound methods for global optimization - I. Theoretical results*, Journal of Global Optimization, Vol. 16, No. 4, pp. 371–392, 2000.
4. T. Csendes, *Generalized Subinterval Selection Criteria for Interval Global Optimization*, Numerical Algorithms, Vol. 37, pp. 93–100, 2004.
5. E. Fitan, F. Messine, B. Nogarde, *The Electromagnetical Actuators Design Problem: a General and Rational Approach*, IEEE Transaction on Magnetics, Vol. 40, No. 3, pp. 1579–1590, 2004.
6. J. Fontchastagner, F. Messine, Y. Lefèvre, *A New Deterministic Global Optimization Algorithm associating with Combinatorial Analytical and Numerical Models to Design Electrical Rotating Machines*, IEEE Transcations on Magnetics, Vol. 43, N. 8, pp. 3411–3419, 2007.
7. E. Hansen, *Global Optimization using Interval Analysis*, Marcel Dekker, Inc. 270 Madison Avenue, New York 100016, 1992.
8. R.B. Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer Academic Publishers, Drodrecht, Boston, London, 1996.
9. F. Messine, *Deterministic Global Optimization using Interval Contraint Propagation Techniques*, RAIRO Operations Research, Vol. 38, No. 4, pp. 277–294, 2004.
10. F. Messine, *A Deterministic Global Optimization Algorithm for Design Problems*, in C. Audet, P. Hansen, G. Savard (editors), Essays and Surveys in Global Optimization, Kluwer, GERAD-Montréal-Canada, pp. 267–294, 2005.
11. F. Messine and J.L. Lagouanelle, *Enclosure methods for multivariate differentiable functions and application to global optimization*, Journal of Universal Computer Science, Vol. 4, No. 6, pp. 589–603, 1998.
12. F. Messine and B. Nogarede, *Optimal Design of Multi-Airgap Electrical Machines: An Unknown Size Mixed-Constrained Global Optimization Formulation*, IEEE Transaction on Magnetics, Vol. 40, N. 3, pp. 3847–3853, 2006.
13. R. E. Moore, *Interval Analysis*, Prentice-Hal, 1966.
14. H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*, Ellis Horwood Ltd, Chichester, England, 1988.
15. O. Shcherbina, A. Neumaier, D. Sam-Haroud, XH. Vu, TV. Nguyen, *Benchmarking global optimization and constraint satisfaction codes*, Lecture Notes in Computer Science, Vol. 2861, pp. 211–222, 2003.
16. B. Tóth and T. Csendes, *Empirical Investigation of the Convergence Speed of Inclusion Function in a Global Optimization Context*, Reliable Computing, Vol. 11, pp. 253–273, 2005.