RAIRO Operations Research
RAIRO Oper. Res. **38** (2004) 277–293
DOI: 10.1051/ro:2004026

DETERMINISTIC GLOBAL OPTIMIZATION USING INTERVAL CONSTRAINT PROPAGATION TECHNIQUES*

Frederic Messine[1]

Abstract. The purpose of this article is to show the great interest of the use of propagation (or pruning) techniques, inside classical interval Branch-and-Bound algorithms. Therefore, a propagation technique based on the construction of the calculus tree is entirely explained and some properties are presented without the need of any formalism (excepted interval analysis). This approach is then validated on a real example: the optimal design of an electrical rotating machine.

Keywords. Interval analysis, Branch-and-Bound, global optimization, pruning/propagation techniques.

Introduction

Deterministic global optimization algorithms based on interval analysis gain actually a new interest, due to their intrinsic quality: the reliable enclosures of the global optimum and also of all the solutions, and due to the performance of the new computers generation. Consequently, the improvements of these methods to solve larger and larger optimization problems, become a real scientific and economic stake.

Constraint propagation techniques were firstly introduced in order to solve the discrete constraint satisfaction problems (discrete CSP): to find the domain in which a given set of constraints is satisfied. These techniques have been more recently extended to the continuous-constraint-satisfaction-problem (continuous CSP) by the use of interval arithmetic [4–6]; then a new adapted formalism was

* *This work was supported by the Laboratoire d'Électrotechnique et d'Électronique Industrielle, Group EM*^{3}*, UMR 5525, Toulouse, France.*

[1] Laboratoire de Mathématiques Appliquées, FRE 2570, Université de Pau et des Pays de l'Adour, UFR Sciences et Techniques, Département d'Informatique, BP 1155, 64013 Pau Cedex, France; e-mail: `Frederic.Messine@univ-pau.fr; messine@leei.enseeiht.fr`

introduced: the box-consistency [8]. A natural extension of these algorithms was to deal with the search of the global optimum of a constrained problem:

$$\begin{cases} \min\limits_{x \in X \subset \mathbb{R}^n} f(x) \\ \qquad g_i(x) \leq 0 \ \ \forall i \in \{1, \cdots, p\} \\ \qquad h_j(x) = 0 \ \forall j \in \{1, \cdots, q\} \end{cases} \tag{1}$$

where X is an hypercube, also named a box, which is the interest domain where the global solution is sought, and f, g_i, h_j are real functions.

The main idea of these global optimization procedures is to consider the following supplementary constraint $f(x) < f_{\min}$ (or $f(x) \leq f_{min}$ for determining all the solutions) where $f_{\min}$ is a current solution which is improved during the iterations of the algorithm [8,19].

In this work, the reverse way is privileged; starting from the standard interval Branch-and-Bound algorithms [7,18] and inserting inside some propagation steps. Thus, all the formalism needed by the previous approach becomes unnecessary and the propagation techniques will be presented without any formalism (only interval analysis). By this way, one focuses solely on the interest of the propagation techniques inside interval Branch-and-Bound algorithms.

In the first section, interval analysis and the classical propagation methods are recalled in the linear case and the extension due to E. Hansen to the non-linear case [2]. Then the following section is dedicated to the study of properties deriving from a propagation technique which is based on a principle of construction and deduction of the calculus tree of the considered constraint. Some works have already been developed in interval-CSP [4,5], and new formalisms have been introduced in [8]. Some modelling language have been implemented in Prolog IV, Numerica [19] in connection with ILOG-solver. In this article, an implementation by overloading operators is completely explained and detailed in the third section. The fourth following section concerns the presentation of the interval Branch-and-Bound algorithm using propagation techniques. The complexity of this new algorithm is discussed there. Then, in the fifth and last section, numerical tests definitely show the great efficiency of such algorithms; a practical example is then considered: the optimal design of a rotating machine with magnetic effects.

1. Interval analysis and classical propagation techniques

1.1. Interval analysis

Interval analysis was introduced by Moore in 1966 [17], to discard numerical errors made by floating point representations and computations. The main idea is to represent a real number by two floating point numbers enclosing it, and then, to perform the computations by replacing this real number by the interval made by the two floating point numbers. Thus, Moore defines operations between intervals.

Let us denote $\mathbb{I}$ the interval compact real set, and for $A \in \mathbb{I}$, $A = [a^L, a^U]$, a^L the *lower bound* of A, and reciprocally a^U the *upper bound*. Interval operations

are defined as follows:

$$\begin{cases} [a,b]+[c,d]=[a+c,b+d], \\ [a,b]-[c,d]=[a-d,b-c], \\ [a,b]\times[c,d]=[\min\{a\times c,a\times d,b\times c,b\times d\},\max\{a\times c,a\times d,b\times c,b\times d\}], \\ [a,b]\div[c,d]=[a,b]\times\left[\frac{1}{d},\frac{1}{c}\right] \text{ if } 0\notin[c,d]. \end{cases} \tag{2}$$

Remark 1.1. Interval arithmetic does not keep all the properties of the classical arithmetic; for example, it is sub-distributive: $A\times(B+C)\subseteq A\times B+A\times C, \forall(A,B,C)\in\mathbb{I}^3$. Furthermore, the subtraction (resp. the division) is not the reverse operation of the addition (resp. the multiplication).

Definition 1.2 (inclusion function)**.** An *inclusion function* denoted by F, is an interval function from $\mathbb{I}$ to $\mathbb{I}$ which encloses the range (also named the direct image) of a considered function f over a box. Therefore, one obtains:

$$\left[\min_{x\in X} f(x),\max_{x\in X} f(x)\right]\subseteq F(X), \forall X\in\mathbb{I}.$$

Theorem 1.3. *The extension into interval of an expression of the considered function f, which consists in – replacing each variable by its corresponding interval, replacing each classical operation by its corresponding interval operation and replacing each univariate classical function, such as* $\sin, \ln$ *by its corresponding interval inclusion function – is an* inclusion function.

This theorem is the fundamental result of the interval analysis [17].

Furthermore, efficient inclusion functions could be constructed, see [9–11, 18] (they could improved the performances of the global optimization algorithm). However, in order to simplify this article, they are not considered here.

1.2. Classical interval propagation techniques

Constraint propagation techniques based on interval analysis permit to reduce the bounds of an initial hypercube (interval vector) by using the implicit dependences between the variables formulated through the constraints.

In this paper, one considers the constraints written as follows:

$$c(x)\in[a,b], \text{ with } x\in X\subset\mathbb{R}^n, \tag{3}$$

where c is a real function which represents the studied constraint, $[a,b]$ is a real fixed interval and X is a real interval compact vector. In order to consider equality constraint, one fixes $a=b$ and, a is fixed to $-\infty$ (numerically one uses the lower representable floating point value) for an inequality constraint.

Linear case:

If the considered constraint is linear: $c(x) = \sum_{i=1}^{n} a_i x_i$. The propagation becomes:

$$X_k := \left(\frac{[a,b] - \sum_{i=1, i\neq k}^{n} a_i X_i}{a_k} \right) \cap X_k, \text{ if } a_k \neq 0, \tag{4}$$

where k is in $\{1, \cdots, n\}$ and X_i is the ith interval component of X.

Non-linear case, E. Hansen Method [2]**:**

If the constraint c is non-linear, E. Hansen proposes to use a Taylor expansion at the first order to produce a linear equation with interval coefficients. A Taylor expansion at the first order can be written as follows:

$$c(x) = c(y) + (x-y).c'(\xi), \forall (x,y) \in X^2, \exists \xi \in \mathring{X},$$

where $(x,y) \in X^2$ and $\xi \in \mathring{X}$ ($\mathring{X}$ represents the open set of the compact hypercube X: a component of $\mathring{X}$ has the following form $]x_i^L, x_i^U[$).

An enclosure of $c'(\xi)$ can be computed with an automatic differentiable code extended to interval, [3, 7, 9]; the computation of an enclosure of c' over X can then be obtained; one notices it by $C'(X)$. Hence, one has: $c'(\xi) \in C'(X)$, with $C'(X) \in \mathbb{I}^n$. Consequently:

$$c(x) \in c(y) + (X-y).C'(X), \forall (x,y) \in X^2.$$

The propagation produces:

$$X_k := \left(\frac{[a,b] - c(y) - \sum_{i=1, i\neq k}^{n} C'_i(X).(X_i - y_i)}{C'_k(X)} + y_k \right) \cap X_k, \text{ if } 0 \notin C'_k(X), \tag{5}$$

where $C'_k(X)$ represents the kth interval component of the vector $C'(X)$ which is an enclosure of the gradient over the box X. When $0 \in C'_k(X)$ no propagation is done. This equation is satisfied for all $y \in X$; generally, one fixes y to the middle of the box X: $y_k = \frac{x_k^L + x_k^U}{2}, \forall k \in \{1, \cdots, n\}$.

Remark 1.4. Hansen proposes in [2] other propagation techniques by solving the so-obtained linear system with interval coefficients by an interval Gauss-Seidel algorithm and also by using the order two of the Taylor expansions.

These two procedures may improve the efficiency of propagation techniques, nevertheless it is not clear what their effects are in classical interval Branch-and-Bound algorithm, and consequently, it is not discussed in this article.

2. Constraint propagation technique based on calculus trees

This propagation technique is based on the construction of the calculus tree of the considered constraint. All the partial computations are stored in each node of the tree and then, one propagates the constraint through the tree, from the root to the leaves.

2.1. Constraint propagation algorithm

Algorithm 2.1 (constraint propagation algorithm)**.**

1. Construction of the calculus tree of the considered constraint c over the box X:
   - *the* **leaves** *are the variables of the problem:* x_i *and its corresponding interval* X_i*, or some constant real parameters;*
   - *on each* **node***, the performed operation is stored with the partial computation; for example:* $X_1 + X_3$*;*
   - *the root is the last computation.*
2. Going down through the tree from the root to the leaves. At each node, one tries by deduction to reduce the partial result computed at the first step and then, the leaves (variables) could be pruned; for example: $X_1 + X_3 = [a, b]$ implies $X_1 := ([a, b] - X_3) \cap X_1$ and $X_3 := ([a, b] - X_1) \cap X_3$.

A detailed implementation of this algorithm is presented in the following section; all the intermediate computations of step 2 are clearly explained. In order to fully understand how the Algorithm 2.1 works, let's consider the following example:

Example 2.2. $2x_3x_2 + x_1 = 3$, with $x_i \in X_i = [1, 3], \forall i \in \{1, 2, 3\}$. The propagation can be seen in Figure 1.

The first step of the Algorithm 2.1 is represented on the left of the scheme Figure 1, and the second step (the propagation phase) on the right. On each node of the construction phase, the partial results are computed and stored, as represented in the left part of Figure 1. Then, the propagation phase (in the right part of Fig. 1) goes down through the tree from the root to the leaves; this procedure makes possible to improve the partial results until the variables (the leaves of the tree). These computations are denoted on the right of each node and each variable. Hence for the variables, an intersection with its initial interval is performed; the results is denoted below the variables. On this Example 2.2, this technique of propagation permits to prove in one step, that the unique solution which satisfies the constraint is $x_1 = x_2 = x_3 = 1$.

2.2. Properties

In this paragraph, some properties are explained in order to show some theoretic interests of Algorithm 2.1.

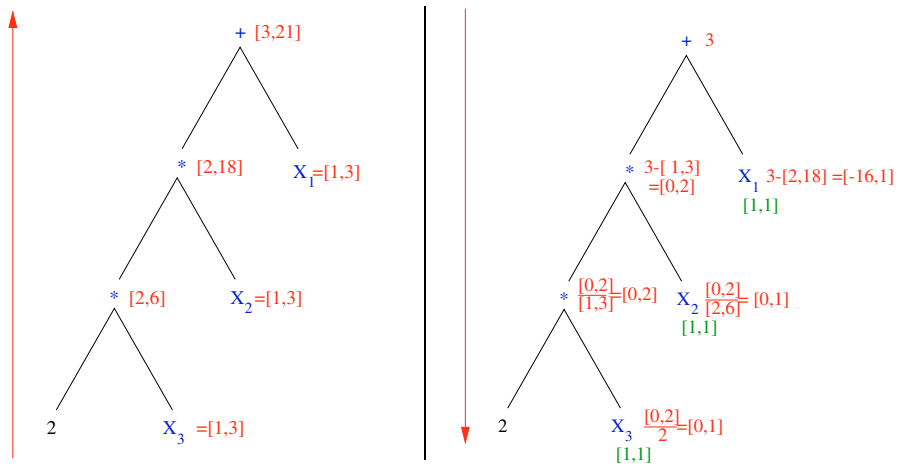

FIGURE 1. Scheme of the use of Algorithm 2.1.

For all these properties, let us denote by Z the box resulting from the propagation of the constraint $c(x) \in [a, b]$ over a considered hypercube X using Algorithm 2.1 and by C the inclusion function of c, defined by using interval analysis [17].

Proposition 2.3. *If x belongs to Z then $c(x)$ belongs to $C(Z)$ and reciprocally, if $x \in X$ and $c(x) \in C(Z)$ then x belongs to Z.*

Proof.

- It is obvious that $x \in Z \Rightarrow c(x) \in C(Z)$, by definition of interval inclusion function, see Definition 1.2.
- The reciprocity is not so evident and is not true in the general case for inclusion functions. However, Z comes from a propagation of the constraint $c(x) \in [a, b]$, and that implies then the following property.

  Algorithm 2.1 uses the calculus tree of an expression of the function c. Therefore by using this tree, the propagation can be expressed for each variable which occurs in the expression of c, as a function depending on the other variables and on the interval $[a, b]$. Let us denote this function by c_i^{-1} for the variable x_i. It is possible that several functions c_i^{-1} exist depending on the expression of c and also of the calculus tree. Therefore, each variable used in the computation of c has the obvious property that: $x_i = c_i^{-1}(d, x)$ where $d = c(x)$ and $d \in [a, b]$. With the expressions of the c_i^{-1} functions, one can construct corresponding inclusion functions by using interval analysis, [17]; they are denoted by C_i^{-1} for the corresponding variable x_i; if there exists many functions c_i^{-1} for one considered variable x_i, one can choose one of them or one can consider the intersection of all of them which is obviously more efficient. Hence, Algorithm 2.1

gives:

$$Z_i \longleftarrow X_i \cap C_i^{-1}([a,b],X).$$

It comes that $x \in Z$ with the property of the inclusion functions constructed by interval analysis C_i^{-1}; because $d \in [a,b]$, $x \in X$ and therefore $c_i^{-1}(d,x) \in C_i^{-1}([a,b],X)$, this involves that $x_i \in Z_i$ for every propagated variables x_i and else $Z_i = X_i$, by definition (for a non-propagated variable x_i). □

Proposition 2.4. *$C(Z) \subseteq C(X)$ and $C(X) \cap [a,b] \subseteq C(Z)$.*

Proof.

- It is evident that $C(Z) \subseteq C(X)$ because $Z \subseteq X$ and by the definition of the interval inclusion functions [17].
- By doing propagation, one can express Z_i by:

$$Z_i \longleftarrow X_i \cap C_i^{-1}([a,b],X)$$

where C_i^{-1} corresponds to the interval inclusion function introduced in the proof of the above Proposition 2.3 and is extended to the case where the variable x_i does not occur in the expression of c: $C_i^{-1}([a,b],X) \longleftarrow [-\infty,\infty]$ and to the case where some occurrences of x_i appear in the expression of c: $C_i^{-1}([a,b],X)$ will denote the intersection of all the distinct computations. Furthermore, one can generally denote $Z = X \cap C^{-1}([a,b],X)$.

Hence, one obtains:

$$\begin{aligned} C(X) \cap [a,b] &\subseteq C(C^{-1}(C(X) \cap [a,b],X)), \\ &\quad \text{by inclusion of the functions } C \text{ and } C^{-1} \\ &\subseteq C(C^{-1}(C(X),X) \cap C^{-1}([a,b],X)). \end{aligned}$$

However, $C^{-1}(C(X),X) = X$, because $X \subseteq C^{-1}(C(X),X)$ and when the propagation is done an intersection is performed with the second term (here X) and then $C^{-1}(C(X),X) \subseteq X$. Hence, one has the result:

$$\begin{aligned} C(X) \cap [a,b] &\subseteq C(C^{-1}(C(X),X) \cap C^{-1}([a,b],X)) \\ &\subseteq C(X \cap C^{-1}([a,b],X)) = C(Z). \end{aligned}$$ □

Theorem 2.5 (validation of Algorithm 2.1)**.** *It does not exist a point x in X, which satisfies a considered constraint c, such that x does not belong to the hypercube Z resulting to the propagation of the constraint c over the box X and using Algorithm 2.1.*

Proof. Let's assume that $x \in X$ and $x \notin Z$ such that $c(x) \in [a,b]$.

Hence, from Proposition 2.3, one has: $c(x) \notin C(Z)$. Furthermore, from Proposition 2.4, $c(x) \notin C(X) \cap [a,b] \subseteq C(Z)$, but $c(x)$ is in $C(X)$ by definition of the

inclusion functions, and therefore, this implies that $c(x)$ does not belong in $[a, b]$ which contradicts the hypothesis. □

Remark 2.6. This Theorem 2.5 proves that Algorithm 2.1 permits to find correct bounds, in the inclusion sense.

Proposition 2.7. *From the two above Propositions 2.3, 2.4 and from Theorem 2.5, one obtains the four following properties:*

1. *the propagation with* $C(X) \cap [a, b]$ *is sufficient;*
2. *if* $C(X) \cap [a, b] = \emptyset$ *then* $Z = \emptyset$*;*
3. *if* $C(X) \subseteq [a, b]$ *then* $Z = X$*;*
4. *if* $C(Z) \subseteq [a, b]$ *then* $c(x) \in [a, b], \forall x \in Z$.

Proof.

1. It is sufficient to propagate $C(X) \cap [a, b]$ because from Theorem 2.5 $\nexists z \in Z \subseteq X$ such that $c(z) \in [a, b]$ and $c(z) \notin C(X)$. Therefore, $\forall z \in Z \subseteq X$, one has $c(z) \in [a, b] \cap C(X)$.
2. From the definition of the propagation and from the above property 1, one has $Z = X \cap C^{-1}([a, b], X) = X \cap C^{-1}(C(X) \cap [a, b], X) = X \cap C^{-1}(\emptyset, X) = \emptyset$.
3. From Proposition 2.4, one has $C(X) \cap [a, b] \subseteq C(Z)$, hence this implies that $C(X)(= C(X) \cap [a, b]) \subseteq C(Z)$. Furthermore, by definition of the propagation and of the inclusion functions, $Z \subseteq X$ and then, $C(Z) \subseteq C(X)$. Therefore, $C(X) = C(Z)$. From Proposition 2.3, one has the result $Z = X$.
4. It is obvious by the definition of inclusion functions because, for all $z \in Z$, $c(z)$ is in $C(Z)$ and therefore $c(z)$ is in $[a, b]$. □

Remark 2.8. The previous properties 2 and 3 eventually permit to improve the convergence of Algorithm 2.1.

3. Detailed implementation of Algorithm 2.1

This method is based on the definition of a new "object" and on the fact that the operations and the classical univariate functions (such as sin or ln for example) can be overloaded. That implies the utilization of oriented object languages, such as Ada, Fortran 90 and 95, and C++.

3.1. Definition of the new object

. Object:

TYPE Node
`Value of the interval,`
`The number of the variable`: 0 if it is an intermediate computation,
`Type of operation`: $+, -, *, /, v$(variable), c(constant), i(integer variable),
`left` : Pointer on left node,
`right` : Pointer on right node; NULL for a univariate function (ln).
END TYPE

. operations: $+, -, \times, \div, \ln, \cdots$ (construction of the calculus tree);
. comparisons: $\leq, =, <, >, \geq$ (propagation of the constraint through the tree, this procedure can generate a reduction of the considered box).

In order to simplify the following subsections, one uses these notations for a considered object O: $O.Val$ for the interval value of the node, $O.num$ for the number of the corresponding variables: i for x_i, $O.op$ for the operations: $+, -, \times, \div$, and the fact that a variable or a constant is considered, $O.left$ to have an access to the left following node in the tree and $O.right$ for the right node; it is fixed to $NULL$ if a univariate function is considered.

3.2. Initialization of the leaves of the calculus tree

The initialization phase begins with the considered box X.

`FOR` i `from 1 to` n `DO`
`Creation of a new pointer on node:` Y_i
$Y_i.Val := X_i$
$Y_i.num := i$
$Y_i.op :=$ "v" or "i" if $i \in E$
$Y_i.left :=$ `NULL`
$Y_i.right :=$ `NULL`
`END DO`

where E represents the set of the indices of the integer variables. This involves a different propagation. For example, if at the end of a propagation $Z_i = [1.2, 3.4]$ is obtained and if $i \in E$ then $Z_i = [2, 3]$.

Remark 3.1. Consequently, mixed-constrained optimization problems could then be considered: real and integer variables, but also logical and categorical variables [13, 14].

3.3. Definition of the overloaded operations

This phase will be automatically performed by overloading the classical operations: $+, -, \times, \div$, and also classical real functions; as for example $\ln x$, $\sqrt{x}$, e^x, $\sin x$.

3.3.1. *Binary operations*

Let us denote by $\Diamond$ one of the classical binary operations: $+, -, \times, \div$.

$X_i \Diamond X_j$:
  `Creation of the result pointer:` $RESULT$
  $RESULT.Val := X_i.Val \Diamond X_j.Val$
  $RESULT.num := 0$
  $RESULT.op :=$ "$\Diamond$"
  $RESULT.left := \&X_i$
  $RESULT.right := \&X_j$

where $\&X$ denotes the memory address of the interval variable X.

The operations between an object Node and a constant value must also be overloaded just by creating the following new object Node, and by performing the appropriated operation:

`Creation of the pointer over a constant` k: K
  $K.Val := [k, k]$
  $K.num := 0$
  $K.op :=$ "c"
  $K.left :=$ `NULL`
  $K.right :=$ `NULL`

3.3.2. *Classical real functions*

Let us denote by $u(x)$, one of the classical most commonly used functions, as for example: $\ln x$, $\sin x$, $\sqrt{x}$, and $U(x)$ the corresponding extension into interval of an expression of $u(x)$, [17].

$u(X_i)$: `Creation of the result pointer` $RESULT$
  $RESULT.Val := U(X_i.Val)$
  $RESULT.num := 0$
  $RESULT.op :=$ "u"
  $RESULT.left := \&X_i$
  $RESULT.right :=$ `NULL`

3.3.3. *Exponent function*

A particular attention must be paid to this function.

X_i^n: `Creation of the resulting pointer` $RESULT$
  $RESULT.Val := X_i.Val^n$
  $RESULT.num := 0$
  $RESULT.op :=$ "pn"

$RESULT.left := \&X_i$
$RESULT.right := \&K$

where the pointer on the constant n must be, *a priori*, declared using the above definition.

3.4. Propagation of the considered constraint

In this paper, only the general case, defined at the previous section, is considered: $c(x) \in [a, b]$; the equality constraint $c(x) = b$ amounts obviously to this case: $c(x) \in [b, b]$ and for inequality constraints $c(x) \leq b$, one considers: $c(x) \in]-\infty, b]$.

Remark 3.2. In order to perform propagations using this implementation, it is simpler to only consider a constraint defined by: $c(x) \in [a, b]$.

By overloading the operator "=" between one node and one interval, the calculus tree will be propagated from the root to the leaves.

Algorithm 3.3 (propagation algorithm)**.** *Let us denote by u a univariate real function, by u^{-1} its inverse function and by U and U^{-1} their classical extension into interval functions* [2, 17, 18].

```
CASE on the type of operation:  (RECURSIVE CALL to :=)
    " + ": right node value := [a,b]− left node value,
       left node value := [a,b]− right node value.
    " − ": right node value := left node value −[a,b],
       left node value := [a,b]+ right node value.
    " × ": right node value := [a,b]÷ left node value,
       left node value := [a,b]÷ right node value.
    " ÷ ": right node value := left node value ÷[a,b],
       left node value := [a,b]× right node value.
    "pn": Let us denote by n the constant value of the
        right node,
          IF n is odd THEN
             left node value := [a,b]^(1/n).
          ELSE let us denoting V the left node value
             IF v^L ≥ 0 THEN
                left node value := [a,b]^(1/n).
             ELSE IF v^L ≤ 0 THEN
                left node value := −[a,b]^(1/n).
             ELSE (0 ∈ V)
                left node value := [−b^(1/n), b^(1/n)].
             END IF
          END IF
    "u": left node value := U^(−1)([a,b]).
```

"v": $X_{\texttt{result number}} \longleftarrow X_{\texttt{result number}} \cap [a,b]$; the recursive procedure is stopped (for "i", a particular rounded intersection is defined).
"c": the recursive procedure is stopped.
END CASE

Remark 3.4. All the intermediate results can be improved only by performing an intersection of the computed value with its previous value, which is calculated at the first step of Algorithm 2.1. If it considered, the recurrence can be stopped whether an intermediate result gives an empty interval.

Remark 3.4 permits to improve Algorithm 3.3. Nevertheless, it is not implemented by now in our version of Algorithm 3.3.

Remark 3.5. This technique could also work with control structure as loops, for example. However, if a part of the computations are performed by a subroutine, attention must be paid in order to construct the correct corresponding calculus tree.

4. Interval Branch-and-Bound algorithm

The main idea is to insert some interval propagation techniques inside classical interval Branch-and-Bound algorithms [9, 18].

Algorithm 4.1 (interval Branch-and-Bound algorithm).

1. $X :=$ initial hypercube in which the global minimum is sought, $X \subseteq \mathbb{R}^n$,
2. $f_{\min} := +\infty$, denote the current minimum,
3. $\mathcal{L} := (+\infty, X)$,
4. Extract from $\mathcal{L}$ the element which has the smallest lower bound,
5. Bisect the considered box normal to a direction: V_1, V_2, - choice of the edge which has the maximal length -,
6. FOR j from 1 to 2 DO
   (a) Compute $v_j :=$ lower bound of f over V_j,
   (b) Pruning of V_j by **CONSTRAINTS PROPAGATION**,
   (c) IF V_j is not empty THEN
   - Compute all the lower and upper bounds of all the constraints over V_j,
   - IF $f_{\min} \geq v_j$ and no constraint is unsatisfied THEN
     - Insert (v_j, V_j) in $\mathcal{L}$,
     - $f_{\min} := \min\{f_{\min}, f(m)\}$, where m is the middle of V_j, if and only if m satisfies all the constraints,
     - IF $f_{\min}$ is modified THEN discard in $\mathcal{L}$ all the couples (z, Z) such that $z > f_{\min}$.
   - END IF,

7. IF $f_{\min} < \min_{(z,Z)\in\mathcal{L}} z+\epsilon_f$ and the largest box remaining in $\mathcal{L}$ is smaller than ϵ THEN STOP. ELSE return to step 4.

Because Algorithm 4.1 stops when the global optimum is enclosed with a given accuracy ϵ_f, and because all the sub-boxes remaining in the list are small enough, at least, one global solution is found: $f_{\min}$ and all the solutions are in the union of all the elements of the list $\mathcal{L}$.

For more detailed descriptions about this Algorithm 4.1, refer to [2, 7, 9, 18]. Attention must be paid at step 5, because the variables can be non-homogeneous, as in physic problems; in order to take into account this difficulty, see [1, 9, 15], to have interesting ways to bisect the box.

Remark 4.2. This program is actually used by the Electroactive Machines and Mechanisms Group of the Laboratoire d'Électrotechnique et d'Électronique Industrielle, for the optimal design of new types of actuators and electrical motors [1, 13–15].

Theorem 4.3. *An overestimation in the worst case of the number of iterations of Algorithm 4.1 (from step 4 to step 7) is given by:* $(\frac{\max_{i\in\{1,\dots,n\}} x_i^U - x_i^L}{\epsilon})^n$. *If the considered problem owns q linear equality constraints which are linearly independent, and if Algorithm 2.1 is used at step 6(b), then an overestimation of the number of loops from step 4 to step 7 becomes* $(\frac{\max_{i\in\{1,\dots,n\}} x_i^U - x_i^L}{\epsilon})^{n-q}$. *Therefore, even if some computations are added (Algorithm 2.1), the exponential complexity of Algorithm 4.1 can strongly be reduced.*

Proof. Consider one equality constraint: $c(x) = \sum_{i=1}^n a_i x_i = b$. At each step of the main loop, the box X is bisected in two parts V and W following a direction k. Thus, if $a_k \neq 0$, the propagation of c must reduce V and W; let's assume that $b \in C(V)$ (else V is discarded).

It is easy to understand that Algorithm 2.1 works like the general case (4) presented in the section 1: $V_j := (\frac{b-\sum_{i=1,i\neq j}^n a_i X_i}{a_j}) \cap V_j$, if $a_j \neq 0$, where j is in $\{1, \cdots, n\}$. However, for all $j \neq k$, V_j is equal to X_j, which has already been propagated (except for the first iteration). Hence, because $V_k = [x_k^L, \frac{x_k^L + x_k^U}{2}]$, all the interval variables V_j will be reduced (for all $j = 1, \cdots, n$, $j \neq k$ and $a_j \neq 0$).

Then, this process works as if a variable was computed from the others; it is in fact the most efficient. Hence, an overestimation of the number of iterations of the main loop is reduced: the exponent decreases about 1.

This proof can easily be extended when q linear equality constraints are considered, if and only if these constraints are linearly independent. Furthermore, this theorem can also be extended to the case where c is non linear, but with at least one variable which can be deduced from the others: $x_k := c_k^{-1}(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n)$. That is the reason why formal modeling languages are introduced: Prolog IV and also Numerica [19]. □

Remark 4.4. Theorem 4.3 shows that constraint propagation techniques make possible the reduction of the exponent of the exponential complexity of the considered problems using the constraints.

Remark 4.5. In this work, Algorithm 4.1 use a bisection phase at step 5. (It is the most classical way to divide a box.) Therefore, the two sub-boxes V_1, V_2 and the father box X differ from only one variable. But, a propagation technique has already been applied on the box X and then, it is easy to understand that only one iteration of propagation is enough in step 6(b) of Algorithm 4.1. That was tested and confirmed on the real example presented in the following section.

Of course, if other multi-section techniques are used in step 5 of Algorithm 4.1, that can involves some interests in the way to perform the propagation.

Remark 4.5 shows that on this kind of Branch-and-Bound Algorithms 4.1 when a bisection is used at step 5, it is not necessary to iterate the propagation techniques over the same box. Therefore, considering global optimization problems, the development of more efficient propagation algorithms does not seem to have any interest except in constraint satisfaction problems [4–6]. Other similar global optimization algorithms are directly based on the extension of the CSP-techniques [8,19]. They also have a great interest; however, they need the understood of a formalism based on box consistency.

5. Application to the design of an electrical machine

This electrical machine has already been designed by deterministic global algorithm [15]; nevertheless the propagation has been performed by hand-computations for only a few variables: $x_i := c_i^{-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. For the last few years, a lot of electromechanical actuators have been designed using Algorithm 4.1.

This real example is taken into account in this paper because it allows to show perfectly the great efficiency of (simple) propagation techniques inside classical interval Branch-and-Bound Algorithm 4.1.

5.1. Analytical model

The complete analytical model is detailed in [15, 16]. In this paper, only the associated optimization problem is presented.

$$\left\{\begin{array}{ll} \min\limits_{x\in X\subset\mathbb{R}^{10}} & \pi\beta l_a \frac{D}{\lambda}(D-2e-l_a) \\ & \Gamma_{em} = \dfrac{\pi}{2\lambda}(1-K_f)\sqrt{k_r\beta E_{ch}E}D^2(D+E)B_e \\ & E_{ch} = AJ_{cu} = k_r E J_{cu}^2 \\ & K_f = 1.5p\beta\dfrac{e+E}{D} \\ & B_e = \dfrac{2l_aP}{D\ln\left(\frac{D+2E}{D-2(l_a+e)}\right)} \\ & C = \dfrac{\pi\beta B_e}{4pB_{iron}}D \\ & p = \dfrac{\pi D}{\Delta_p} \end{array}\right. \tag{6}$$

where the ten variables of the problem are: $D(m) \in [0.01, 0.5]$ *the bore diameter*, $\lambda \in [1, 2.5]$ which is *the diameter over length ratio*, $l_a(m) \in [0.003, 0.05]$ *the thickness of the permanent magnets*, $E(m) \in [0.001, 0.05]$ *the winding thickness*, $C(m) \in [0.001, 0.05]$ *the thickness of yoke*, $\beta \in [0.8, 1]$ *the polar arc factor*, $B_e(T) \in [0.1, 1]$ *the magnetic field in the air gap*, $J_{cu}(A/m^2) \in [10^5, 10^7]$ *the current areal density*, $K_f \in [0.01, 0.3]$ *a semi-empiric magnetic leakage coefficient* (established by numerical simulations) and $e(m) \in [0.001, 0.005]$ *the thickness of the mechanical air gap.*

The other parameters are fixed: $p = 4$ *the number of pole pairs*, $k_r = 0.7$, $B_{iron} = 1.5$ T *the magnetic field in the iron*, $E_{ch} = 10^{11}$ A/M, $\Gamma_{em} = 10$ N.m *the electromagnetical torque*, $P = 0.9$ T *the magnetic polarization* and $\Delta_p = 0.1$ m *the polar step.*

This is a problem of optimal dimensioning of rotating slotless machines with permanent magnets, [15]. The considered criterion to be minimized for this example, is the magnet volume (in fact, the most expensive component of the actuator). The global solution must satisfy some technical constraints, as a fixed torque ($\Gamma_{em} = 10$ N.m). This comes from an analytical model which takes some assumptions to analytically solve the Maxwell partial differential equations, see [1,15,16].

5.2. Numerical results

These numerical tests have been performed on a 800 MHz HP bi-processors server with 512 Mo of RAM; these tests have been done several times, in order to obtain comparable CPU-times for each method. These algorithms have been implemented with Fortran 90.

On this real example, one can note that all the propagation techniques permit to strongly decrease the CPU-time and the number of iterations of these kind of interval Branch-and-Bound Algorithms 4.1. The propagation technique based on the calculus tree and entirely detailed in this paper is the most efficient (on this example) and permits to obtain the solution in a record time: only 0.5 second for such an optimization problem! Compared to the hour and half necessary for the classical method, the gain is impressive. Consequently, these propagation techniques become unavoidable to solve such constrained optimization problems.

However, one can notice a difference between the global solutions obtained. In fact, the global solution given by the classical Algorithm 4.1 directly used without propagation, seems more efficient than those produced by performing some propagation steps. This is due to the fact that the constraints are satisfied with more accuracy when propagation techniques are used.

If the designer wants to introduce a tolerance on these constraints, it is possible, by modifying the equality constraints $h_j(x) = 0$ by $h_j(x) \in [-\epsilon_j, \epsilon_j]$ and the inequality constraints $g_j(x) \leq 0$ by $g_j(x) \in]-\infty, \epsilon_j]$ (where all the $\epsilon_j > 0$ are given by the user of Algorithm 4.1). In that case, the same solutions are obtained (denoted by "with an ε", $\varepsilon = (3 \times 10^{-2}, 10^{10}, 10^{-2}, 10^{-2}, 10^{-3}, 2 \times 10^{-3})$); one can note that the CPU-time and the number of iterations become less efficient but remain still interesting: only 17 seconds compared to one hour and half!

TABLE 1. Dimensioning of a rotating slotless machine, minimization of the magnet volume.

Algorithm 4.1 + Propagation	Minimum	Number of iterations	CPU-times
Without Propagation	6.20×10^{-4}	407 838	$\simeq$1 h 35
Hand-computed	6.76×10^{-4}	1340	1.9 s
Hansen technique	6.79×10^{-4}	609	41.5 s
Algorithm 2.1	6.81×10^{-4}	120	0.5 s
Algorithm 2.1 +Hansen technique	6.80×10^{-4}	43	2.7 s
Algorithm 2.1 with an ε	6.21×10^{-4}	4641	16.8 s

The Hansen method does not give an efficient result (around 30 mns). Nevertheless, Hansen propagation technique must be associated with a interval Gauss-Seidel algorithm to become really efficient, [2]; this is not implemented in this work.

Remark 5.1. Considering only one constraint, the CPU-time of one propagation of the Hansen method is approximately eight times superior to the CPU-time of one iteration of Algorithm 2.1.

Other comparisons of some propagation methods (without optimization) can be found in [4, 12], and one application of such techniques can be found in [5].

Remark 5.2. Considering this optimal design problem, the global solutions had not been found using classical optimization tools: Lagrangian augmented or SQP algorithms [16]. That revealed the great interest of this deterministic global optimization approach [15]. Furthermore, in order to solve more general design problems, some extensions of Algorithm 4.1 has already been implemented to deal with mixed-constrained optimization problems: such a method must deal with real, integer, logical and categorical variables [13, 14].

CONCLUSION

In this article, a propagation method based on the construction of the calculus tree is entirely explained without the need of any formalism: including corresponding properties and a detailed implementation. Its introduction inside classical Branch-and-Bound algorithms shows a great interest through a real example: the optimal design of an electrical slotless rotating machine with permanent magnets. Thus, propagation techniques integrated inside interval Branch-and-Bound algorithm yield new perspectives for the generalization of the utilization of such deterministic global optimization methods.

References

[1] E. Fitan, F. Messine and B. Nogarede, The Electromagnetical Acuator Design Problem: A General and Rational Approach. *IEEE T. Magn.* **40** (2004).

[2] E. Hansen, *Global Optimization Using Interval Analysis*. Marcel Dekker, Inc. 270 Madison Avenue, New York 10016 (1992).

[3] J.-C. Gilbert, G. Le Vey and J. Masse, *La différentiation automatique de fonctions représentées par des programmes*. Rapports de Recherche de l'INRIA- Rocquencourt, 1557, Programme 5, Traitement du Signal, Automatique et Productique (1991).

[4] L. Granvilliers, On the Combination of Interval Computating Solvers. *Reliab. Comput.* **7** (2001) 467–483.

[5] L. Granvilliers and F. Benhamou, Progress in the Solving of a Circuit Design Problem. *J. Global Optim.* **20** (2001) 155–168.

[6] L. Jaulin, Interval Constraint Propagation with Application to Bounded-Error Estimation. *Automatica* **36** (2000) 1547–1562.

[7] R.B. Kearfott, *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dordrecht, Boston, London (1996).

[8] O. Lhomme, A. Gotlieb and M. Ruher, Dynamic Optimization of Interval Narrowing Algorithms. *J. Logic Program.* **37** (1998).

[9] F. Messine, *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution de problèmes avec contraintes*. Ph.D. Thesis, Institut National Polytechnique de Toulouse (1997).

[10] F. Messine, Extension of Affine Arithmetic: Application to Global Optimization. *J. Universal Comput. Sci.* **8** (2002) 992–1015.

[11] F. Messine and J.L. Lagouanelle, Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. *J. Univ. Comput. Sci.* **4** (1998) 589–603.

[12] F. Messine, *Méthodes de propagation de contraintes basées sur l'analyse d'intervalles pour l'optimisation globale déterministe*. Rapport interne de recherche du Département Informatique de l'UPPA, R2I01-02, 18 pages (2002). Available on `www.univ-pau.fr/~ messine`

[13] F. Messine, V. Monturet and B. Nogarede, *An Interval Branch and Bound Method Dedicated to the Optimal Design of Piezoelectric Actuators*. Mathematics and Computers in Science and Engineering, ISBN 960-8052-36-X, WSES Press (2001) 174–180.

[14] F. Messine, E. Fitan and B. Nogarede, The Inverse Problem Associated to the Optimal Design of Electromagnetic Actuators: Application to Rotating Machines with Magnetic Effects, in *European Symposium on Numerical Methods in Electromagnetics, Proceedings JEE'02* (2002) 318–323.

[15] F. Messine, B. Nogarede and J.L. Lagouanelle, Optimal Design of Electromechanical Actuators: A New Method Based on Global Optimization. *IEEE T. Magn.* **34** (1998) 299–307.

[16] B. Nogarede, A.D. Kone and M. Lajoie-Mazenc, Optimal Design of Permanent-Magnet Machines Using an Analytical Field Modeling. *Electromotion* **2** (1995) 25–34.

[17] R.E. Moore, *Interval Analysis*. Prentice Hall, Inc. Englewood Cliffs, N.J. (1966).

[18] H. Ratschek and J. Rokne, *New computer methods for global optimization*. ELLIS HORWOOD LIMITED Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB, England (1988).

[19] P. Van Henterbryck, L. Michel and Y. Deville, *Numerica: a Modelling Language for Global Optimization*. MIT Press, Cambridge Mass (1997).