Mind the Gaps: A New Splitting Strategy for Consistency Techniques

Heikel Batnini, Claude Michel, and Michel Rueher

Université de Nice Sophia-Antipolis,
COPRIN project I3S-CNRS/INRIA/CERTIS,
INRIA, 2004 Route des Lucioles,
BP 93, 06902 Sophia-Antipolis, France
Heikel.Batnini@sophia.inria.fr
{cpjm, rueher}@essi.fr

Abstract. Classical methods for solving numerical CSPs are based on a branch and prune algorithm, a dichotomic enumeration process interleaved with a consistency filtering algorithm. In many interval solvers, the pruning step is based on local consistencies or partial consistencies. The associated pruning algorithms compute numerous data required to identify gaps within some domains, *i.e.* inconsistent intervals strictly included in the domain. However, these gaps are only used to compute the smallest approximation of the box enclosing all the solutions. This paper introduces a search strategy, named MindTheGaps, that takes advantage of the gaps identified during the filtering process. Gaps are collected with a negligible overhead, and are used to select the splitting direction as well as to define relevant cutting points within the domain. Splitting the domain by removing such gaps definitely reduces the search space. It also helps to discard some redundant solutions and helps the search algorithm to isolate different solutions. First experimental results show that MindTheGaps significantly improves performances of the search process.

1 Introduction

Many application problems ranging from robotics to chemistry and geometry can be seen as numerical constraint satisfaction problems (NCSPs). A NCSP is defined by a set of variables and a set of nonlinear constraints on the variables. The domain of the variables are closed intervals of real values. Numerical CSPs can be used to express a large class of problems, particularly problems with imprecise data or partially defined parameters. The goal is to find sharp boxes that approximate the solutions. Correct approximations of the solutions can be obtained by interval-based solvers; most of them implement a search algorithm that combines enumeration techniques and local consistencies techniques.

Consistencies techniques over numerical CSPs are derived from finite domains CSPs techniques. The associated filtering algorithms remove from the interval domains some values for which at least one constraint does not hold (inconsistency). In practice, the pruning is limited to a contraction of the bounds of the intervals.

Classical techniques for solving numerical CSPs are based on a branch and prune algorithm. This algorithm interleaves domain pruning and domain splitting, until a given

P. van Beek (Ed.): CP 2005, LNCS 3709, pp. 77–91, 2005.

precision of the domains is reached. The splitting step selects a direction and splits the corresponding interval in several pieces. The standard splitting technique is bisection, which splits the selected domain in its middle.

Among the strategies for selecting the domain to split, the method considered as the most efficient on average is the Round Robin method (RR) : the domains of the variables are processed alternately. However, other domain selection strategies have been proposed. The Largest First (LF) strategy, also called geometric splitting [1], selects first the domain of maximal width. The Maximal Smear (MS) strategy has been introduced by [2] for interval Gauss-Seidel method : the selected domain maximizes the smear function[1] [3], informally speaking, the domain of the variable the projection of which has the strongest slope.

In most interval solvers, the pruning step is based on local consistencies (Hull-Consistency [4,5], Box-consistency [6,7,8]) or stronger consistencies (kB-consistencies [4,9], Bound-consistency [10]). The associated pruning algorithms often identify gaps within some domains, *i.e.*, inconsistent intervals strictly included in the domain. These gaps are only used to compute the smallest approximation of the box enclosing all the solutions.

This paper introduces a search strategy, named MindTheGaps, that takes advantage of the gaps identified by local consistencies filtering algorithms. These gaps are collected with a negligible overhead, and are used to select the splitting direction as well as to define relevant cutting points within the domain. Splitting a domain by removing such a gap definitely reduces the search space. It also helps to discard some redundant solutions and helps the search algorithm to isolate different solutions. If no gap has been found, the branching step is achieved by a standard splitting process combined with classical selection strategies.

In general, chronological backtracking is used to handle the subproblems generated by the splitting step. However, more sophisticated strategies may also be used, as for instance a dynamic backtracking strategy [11]. Note that MindTheGaps is fully compatible with any backtracking technique.

A similar approach has been suggested by Hansen [12,2] for interval Newton method. The search algorithm exploits the gaps identified by Gauss-Seidel steps. This approach has been used by Ratz [1] for handling global optimization problems. Three different box-splitting strategies have been suggested :

- Use only the largest gap to split the box, and generate 2 subproblems [12].
- Use k gaps found in the same domain to split the box and generate $k + 1$ subproblems [1].
- Use at most three gaps in three different domains, and combine the subdomains to generate up to 8 subproblems [2].

We generalize Hansen's approach for all classical consistency filtering algorithms : Hull-consistency, Box-consistency and kB-consistencies. We demonstrate that this approach works well for solving satisfaction problems, that is to say for finding all isolated solutions or solutions spaces.

[1] The smear function of x_k is : $s_k = \max_{1 \leq j \leq m} \{\max \{|\underline{J}_{i,j}|, |\overline{J}_{i,j}|\} w(\mathbf{x}_i)\}$, where $\mathbf{J}_{i,j} = [\underline{J}_{i,j}, \overline{J}_{i,j}]$ is the (i, j)-th entry of the interval extension of the Jacobian matrix of the system.

Hyvönen [13] used the gaps to enforce strong consistency. He proposed an algorithm to enforce union-consistency by combining sets of intervals, but this method is strongly limited by its exponential character. MindTheGaps uses the gaps to guide the solution space exploration, and can thus limit the number of generated gaps. To limit the cost of the management of unions of intervals, we avoid gap identification on trigonometric functions. More precisely, we restrict gap identification to power terms and divisions, which produce at most one gap.

The paper is organized as follows: section 2 briefly describes the notations used in the rest of the paper. Section 3 gives an overview of MindTheGaps. Section 4 describes the extensions of Hull-consistency, Box-consistency filtering algorithms that collect the gaps. Section 5 reports some experimental results on classical benchmarks. Finally, section 6 proposes different extensions of the method to handle stronger consistencies.

2 Notations

Let $\overline{\mathbb{R}}$ be the set of real numbers $\mathbb{R}$ extended to infinites values $\{-\infty, +\infty\}$ and let $\overline{\mathbb{F}} \subset \overline{\mathbb{R}}$ be the subset of reals corresponding to binary floating-point numbers in a given format. A closed interval $\mathbf{x} = [\underline{x}, \overline{x}]$ with $\underline{x}, \overline{x} \in \overline{\mathbb{F}}$ denotes the set of real values x such that $\underline{x} \leq x \leq \overline{x}$. Open intervals will be denoted by $(\underline{x}, \overline{x}) = \{x \in \overline{\mathbb{R}} \text{ s.t. } \underline{x} < x < \overline{x}\}$. $\mathbb{I}$ stands for the set of such intervals and $\cap_{\mathbb{I}}$ denotes the intersection operator over $\mathbb{I}$.

A union of intervals is denoted by $\mathbf{u} = \bigcup \mathbf{u}_{(j)}$, where the subintervals $\mathbf{u}_{(j)}$ are disjoint and sorted by increasing lower bound, *i.e.* $\overline{u}_{(j)} < \underline{u}_{(j+1)}$. The number of subintervals of $\mathbf{u}$ is denoted by $|\mathbf{u}|$. The lower bound (resp. upper bound) of $\mathbf{u}$ is denoted by $\underline{\mathbf{u}}$ (resp. $\overline{\mathbf{u}}$). $\mathbb{U}$ stands for the set of such unions of intervals and $\cap_{\mathbb{U}}$ denotes the intersection operator over $\mathbb{U}$, such that $\mathbf{u} \cap_{\mathbb{U}} \mathbf{v} = \{x \in \mathbb{R} : x \in \mathbf{u} \wedge x \in \mathbf{v}\}$.

Real variables are denoted by x, y and X, Y denote variable vectors whereas $\mathbf{X}, \mathbf{Y}$ denote interval vectors and $\mathbf{U}, \mathbf{V}$ denote vectors of union of intervals. We note $\mathbf{X} = \emptyset$ whenever one of the interval components of $\mathbf{X}$ is empty. The width of an interval $w(\mathbf{x})$ is the positive quantity $\overline{x} - \underline{x}$, while the midpoint $m(\mathbf{x})$ of the interval $\mathbf{x}$ is $(\overline{x} + \underline{x})/2$. $w(\mathbf{X})$ denotes the size of the largest interval component of $\mathbf{X}$.

This paper focus on numerical CSPs defined by $X = (x_1, \ldots, x_n)$, a vector of variables, $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, a vector of associated domains, and $\mathbf{C} = \{c_1, \ldots, c_m\}$, a set of nonlinear constraints. The set of variables of the constraint c is denoted by $\mathbb{V}_c$.

3 MindTheGaps: General Framework

Classical techniques for solving numerical CSPs are based on a branch and prune algorithm (see figure 1(a)). This algorithm interleaves domain pruning and domain splitting until a given precision ω_{sol} of the domains is reached. Prune (line 4) is one of the standard filtering algorithm based on numerical constraint programming consistency techniques: Hull-consistency, Box-consistency or kB-consistencies. In the rest of the paper, Hull-consistency algorithm will be denoted by HCPrune and Box-consistency algorithm by BCPrune. Split (line 9) is a function that selects a splitting direction and splits the corresponding interval. The generated subproblems are added to the set Q. In

```
BranchAndPrune(in :X_0, C, ω_sol out : S)
  %% X_0 = (x_1, ..., x_n)
1: Q ← {X_0} ; S ← ∅
2: while Q ≠ ∅ do
3:   Extract X from Q
4:   X ← Prune(C, X)
5:   if X ≠ ∅ then
6:     if w(X) ≤ ω_sol then
7:       S ← S ∪ X
8:     else
         % Standard splitting process
9:       Q ← Q∪ Split(X)
10:    endif
11:  endif
12: endwhile
13: return S
```

(a) Standard BranchAndPrune algorithm

```
MindTheGaps(in:X_0, C, ω_sol out: S)
  %% X_0 = (x_1, ..., x_n)
1: Q ← {X_0} ; S ← ∅
2: while Q ≠ ∅ do
3:   Extract X from Q
4:   X ← Prune*(C, X, U)
5:   if X ≠ ∅ then
6:     if w(X) ≤ ω_sol then
7:       S ← S ∪ X
8:     else if ∃k s.t. |u_k| > 1 then
         % Gap Splitting
9:       Q ← Q∪ GapSplit(U)
10:    else
         % Standard splitting process
11:      Q ← Q∪ Split(X)
12:    endif
13:  endif
14: endwhile
15: return S
```

(b) MindTheGaps:Overview

Fig. 1. Overall scheme of MindTheGaps

general, bisection is used and the intervals are split in their middle. Different domain selection strategies may be used such as RR, LF or MS (as mentioned in section 1).

In contrast, MindTheGaps (see figure 1(b)) takes advantage of the gaps produced by consistency filtering algorithms. Function Prune* (line 4) collects the gaps generated during the filtering process. The identified gaps are stored in $\mathbf{U}$, which is a vector of union of intervals $(\mathbf{u}_1, \ldots, \mathbf{u}_n)$, such that : $\mathbf{u}_i = \bigcup \mathbf{u}_{i(j)}$, where $\mathbf{u}_{i(j)} = [\underline{u}_{i(j)}, \overline{u}_{i(j)}]$ denotes the j-th sub-domain of x_i. As long as $w(\mathbf{X})$ is larger than some ω_{sol}, MindTheGaps splits at first among a domain that contains at least one gap (line 9). Several heuristics for selecting the domain to split and for choosing the gaps to remove have been explored (see section 5). The splitting is actually done by the function GapSplit which removes one or more gaps from the selected domains, and stores the subproblems in the stack Q. MindTheGaps splits first the domains that contain gaps. If no gap has been found, the standard Split is used (line 11).

4 Local Consistencies and Gaps

Most constraint solvers (*e.g.* IlogSolver [14], Numerica [7], Realpaver [15]) are based on local consistencies (Hull-consistency [4,5], Box-consistency [7,6]). The corresponding filtering algorithms perform a pruning of the domains of the variables by removing values for which some constraints do not hold (inconsistency). This reduction is achieved by narrowing operators which are correct, monotone and contracting functions. The reductions are propagated using the standard interval narrowing algorithm, derived from AC3 [16] (see figure 2).

Prune(in:$\mathbf{C}$, $\mathbf{X}_0$)
% $\mathbf{X}_0 = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
% $\mathbf{C} = \{c_1, \ldots, c_m\}$
1: $Q \leftarrow \mathbf{C}$; $\mathbf{X} \leftarrow \mathbf{X}_0$
2: **while** $Q \neq \emptyset$ **and** $\mathbf{X} \neq \emptyset$ **do**
3: extract c_i from Q
4: $\mathbf{X}' \leftarrow$ Narrow$(c_i, \mathbf{X})$
5: **if** $\mathbf{X} \neq \mathbf{X}'$ **then**
6: $Q \leftarrow Q \cup \{c_j | \exists x_k \in \mathbb{V}_{c_j} \wedge \mathbf{x}_k \neq \mathbf{x}'_k\}$
7: $\mathbf{X} \leftarrow \mathbf{X}'$
8: **endif**
9: **endwhile**
10: **return** $\mathbf{X}$

Fig. 2. Standard interval narrowing algorithm

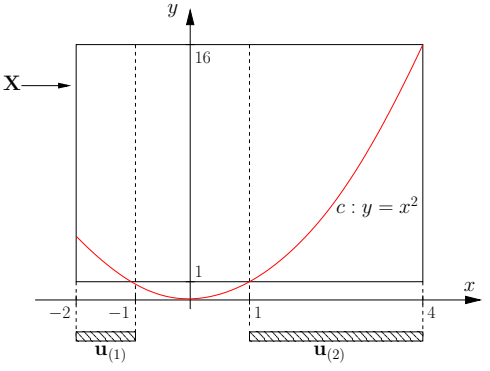

Fig. 3. Approximation of a projection function by a union of intervals

Hull-consistency and Box-consistency are based on this standard interval narrowing algorithm, but they use a specific Narrow function. In this section, we describe extended Narrow functions that collects the gaps identified by the respective consistency. These extended versions, called Narrow$^\star(c_i, \mathbf{X}, \mathbf{U})$, store the gaps identified in a vector of unions of interval $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$. Then we define extended Hull-consistency and Box-consistency filtering algorithms that collect the identified gaps.

4.1 Interval Extensions and Projection Functions

An interval evaluation of a real-valued function f for a given range $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is an interval $\mathbf{y}$ such that : $\underline{y} \leq f(x_1, \ldots, x_n) \leq \overline{y}$, $\forall x_i \in \mathbf{x}_i, 1 \leq i \leq n$. In other words, $\mathbf{y}$ is an interval that contains the values of f, when the values of the unknowns are restricted to the box $\mathbf{X}$. The simplest way to compute $\mathbf{y}$ is to evaluate the *natural* interval extension of f, obtained by substituting all classical mathematical operators (resp. constants, variables) in f by their *basic* interval extension [17].

Example 1. Let $f(x, y) = 2x + xy - 1$ with $x \in [-1, 1], y \in [1, 2]$. The interval evaluation of f for the given ranges is $[2, 2] \otimes [-1, 1] \oplus [-1, 1] \otimes [1, 2] \ominus [1, 1] = [-5, 3]$.

The same principle can be applied to compute a union of intervals that contains the values of f using the *extended* interval extension of the basic mathematical operators [17,2,18,19].

Interval narrowing algorithms use projection functions [8] to prune the domains of the variables. Informally speaking, $\pi_c^{x_k}(\mathbf{X})$ denotes the projection over the variable x_k of the solutions of c when the values of the variables are restricted to the range $\mathbf{X}$. Note that $\pi_c^{x_k}(\mathbf{X})$ may be conservatively approximated either by the smallest enclosing interval denoted by $\Box_{\mathbb{I}}(\pi_c^{x_k}(\mathbf{X}))$, or by the smallest enclosing union of intervals denoted by $\Box_{\mathbb{U}}(\pi_c^{x_k}(\mathbf{X}))$ (see example 2).

Example 2. Consider the constraint $c : y = x^2$ where $x \in [-2, 4]$ and $y \in [1, 16]$ and let $\mathbf{X} = [-2, 4] \times [1, 16]$. Figure 3 shows that the interval approximation of $\pi_c^x(\mathbf{X})$ is $\Box_{\mathbb{I}}(\pi_c^x(\mathbf{X})) = [-2, 4]$, whereas the union approximation of $\pi_c^x(y)$ is $\Box_{\mathbb{U}}(\pi_c^x(\mathbf{X})) = [-2, -1] \cup [1, 4]$.

To limit the cost of the management of unions of intervals, we avoid gap identification on trigonometric functions. More precisely, gap identification is restricted to power terms and divisions whose projections produce at most one gap[2]. The influence of the syntactic form of the constraint over gap identification will be explored in section 5.

4.2 Hull-Consistency and Gaps

Hull-consistency states a local property on the bounds of the domains. A constraint c is Hull-consistent if for any variable x_i of c, there exist values in the domains of all the other variables which satisfy c when x_i is fixed to $\underline{x_i}$ or $\overline{x}_i$. A more formal definition of Hull-consistency can be found in [4].

The basic implementation of Hull-consistency, named 2B-consistency [4,20], decomposes the system of constraints into primitive constraints for which projection functions are easy to compute. The most powerful implementation of Hull-consistency is HC4 [5], based on the narrowing operator HC4revise. This implementation does not require any explicit decomposition of the constraints. All the projection functions are evaluated by traversing a tree-structured representation of the constraints from bottom to top and conversely. Forward propagation evaluates the expression associated to each nodes, using the domains or the values associated to its subtrees. Backward propagation traverses the tree in the opposite direction to reduce the domains of the variables using the projection functions associated to each operator.

We detail now HCNarrow, the narrowing operator used for computing Hull-consistency in HC4. That is to say, HCNarrow corresponds to function Narrow in the generic algorithm of figure 2. Basically, HCNarrow (figure 4(a)) prunes the domain vector $\mathbf{X}$ by applying a constraint narrowing operator on each variable x_k of $\mathbb{V}_c$. This narrowing operator reduces the bounds of the domain of x_k by computing the natural extension of the projection function $\pi_c^{x_k}$. The evaluation of $\pi_c^{x_k}$ by union of intervals is intersected with $\mathbf{x}_k$ (line 2), but the possible gaps are lost during the intersection operation. In fact, they are used only to compute a stronger pruning of $\mathbf{x}_k$. However, the gaps could be collected by replacing the interval intersection operation, $\cap_{\mathbb{I}}$, in HCNarrow (line 2) by the pending operation on union of intervals, $\cap_{\mathbb{U}}$. Actually, it is not necessary to compute the intersection of the union of intervals at each projection, which may be costly. The narrowing operators being contracting functions ($\Phi(\Omega) \subseteq \Omega$), the last projection provides the smallest union of intervals (w.r.t. set inclusion) that approximate $\pi_c^{x_k}$. For this reason, HCNarrow* (see picture 4(b)) maintains a set S of constraint/variable pairs for which gaps have been identified. The effective computation of the gaps is delayed to the end of the propagation step. More precisely, HCNarrow* checks whether the evaluation of the projection function $\pi_c^{x_k}$ produces a gap within the domain $\mathbf{x}_k$ (line 5-9). In this case, the (c, x) is added to set of pairs of constraint/variable S. Otherwise (c, x) is

[2] Note that several gaps may be produced by intersecting the projections of different constraints on the same domain.

HCNarrow(in:c, $\mathbf{X}$) : Interval vector

```
% X = (x_1, ..., x_n)
1: foreach x_k ∈ V_c do
2:    x_k ← □_I(x_k ∩_U □_U(π_c^{x_k}(X)))
   % Possible gaps are lost
3:    if x_k = ∅ then
4:       return ∅
5:    endif
6: endfor
7: return X
```

(a) HCNarrow

HCNarrow*(in:c, $\mathbf{X}$, in-out: S) : Interval vector

```
% X = (x_1, ..., x_n)
1: foreach x_k ∈ V_c do
2:    u ← x_k ∩_U □_U(π_c^{x_k}(X))
3:    if u = ∅ then return ∅
4:    else   % Mind the gap
5:       if |u| > 1 then
6:          S ← S ∪ (c, x_k)
7:       else
8:          S ← S \ (c, x_k)
9:       endif
10:      x_k ← □_I(u)
11:   endif
12: endfor
13: return X
```

(b) HCNarrow*

Fig. 4. Hull-consistency narrowing function and its variant which collects the gaps

deleted from S to handle the case where a gap has been previously identified but pushed out of the domain during the propagation step.

Let HCPrune$^+$($\mathbf{C}$,$\mathbf{X}$,S) be the algorithm Prune (see figure 2), in which the call to the narrowing function have been replaced by HCNarrow*(c_i,$\mathbf{X}$,S). HCPrune$^+$ enforces hull-consistency over the box $\mathbf{X}$ and fills the set S with pairs of constraint/variables for which gaps have been identified.

Then, for each pair (c, x_k) of S, HCPrune* retrieves the gaps by intersecting $\mathbf{u}_k$ with the evaluation by union of intervals of the projection $\pi_c^{x_k}$ (see figure 5).

4.3 Box-Consistency and Gaps

Box-consistency [6,7] is a coarser approximation of arc-consistency than Hull-consistency, but it achieves a stronger pruning in practice [8]. Moreover, Box-consistency tackles some dependency problems when variables have multiple occurrences in the same constraint. A constraint c is Box-consistent if for any variable x_i of $\mathbb{V}_c$, the bounds of $\mathbf{x}_i$ satisfy the unary constraint obtained by replacing each occurrence of a variable x_j other that x_i by the constant interval $\mathbf{x}_j$. A more formal definition of Box-consistency can be found in [6,7].

Box-consistency generates a set of univariate functions which can be tackled by numerical method such as Interval Newton [2]. The pruning consists in finding the leftmost and the rightmost quasi-zero[3] of these univariate functions. Let BCNarrow be the narrowing function used to compute Box-consistency. BCNarrow(c,$\mathbf{X}$) prunes the domain of each variables of c until c is Box-consistent. For each variable x_k of c, an interval univariate function $\mathbf{f}_{\mathbf{x}_k}$ is generated from c by replacing all the variables but x_k by their interval domain. Then, the pruning consists in finding the leftmost quasi-zero and the rightmost quasi-zero of $\mathbf{f}_{\mathbf{x}_k}$. This narrowing is achieved on the lower bound

[3] A quasi-zero of the interval function $\mathbf{f}$ is an interval vector $\mathbf{X}$ such that $0 \in \mathbf{f}(\mathbf{X})$.

HCPrune*(in:$\mathbf{C}$, in-out: $\mathbf{X}$, $\mathbf{U}$)

% $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
% $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$
1: $S \leftarrow \emptyset$
2: HCPrune$^+$($\mathbf{C}$,$\mathbf{X}$,S)
3: **if** $\mathbf{X} \neq \emptyset$ **then**
% Collect the gaps
4: **foreach** $(c, x_k) \in S$ **do**
5: $\mathbf{u}_k \leftarrow \mathbf{u}_k \cap_{\mathbb{U}} \square_{\mathbb{U}}(\pi_c^{x_k}(\mathbf{X}))$
6: **endfor**
7: **endif**

Fig. 5. HCPrune* enforces Hull-consistency and collects the gaps

BCNarrow(in:c, $\mathbf{X}$) : Interval vector

% $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
1: **foreach** $x_k \in \mathbb{V}_c$ **do**
2: $\mathbf{x}_k \leftarrow$ LeftNarrow$(\mathbf{f}_{\mathbf{x}_k}, \mathbf{f}'_{\mathbf{x}_k}, \mathbf{x}_k)$
3: $\mathbf{x}_k \leftarrow$ RightNarrow$(\mathbf{f}_{\mathbf{x}_k}, \mathbf{f}'_{\mathbf{x}_k}, \mathbf{x}_k)$
4: **if** $\mathbf{x}_k = \emptyset$ **then**
5: **return** $\emptyset$
6: **endif**
7: **endfor**
8: **return** $\mathbf{X}$

Fig. 6. Box-consistency narrowing function

by LeftNarrow (see figure 7(a)) and on the upper bound RightNarrow (see [6] for a detailed description of these algorithms). These functions are based on MonoNewton, which prunes the domain of a variable $\mathbf{x}$ w.r.t. a constraint c using the classical univariate interval Newton algorithm. Whenever $\mathbf{x}$ is reduced less than a given ϵ, a splitting process is applied to ensure that $\underline{x}$ is a quasi-zero of $\mathbf{f}_\mathbf{x}$. LeftNarrow* (see figure 7(b)) collects the gaps identified by the Box-consistency narrowing operator.

The point is that the call of MonoNewton (line 3 of LeftNarrow) produces gaps in two different ways:

1. If the right bound of the current interval domain $\mathbf{x}$ is reduced by MonoNewton, the removed interval $([\overline{x}', \overline{x}])$ does not satisfy c.
2. By the interval newton method, MonoNewton, itself.

To explain how MonoNewton may produce gaps, let us recall the definition of the interval Newton method :

$$\begin{aligned} \mathbf{x}^{(0)} &= \mathbf{x} \\ \mathbf{x}^{(n+1)} &= N(\mathbf{f}, \mathbf{f}', \mathbf{x}^{(n)}), \\ \text{where } N(\mathbf{f}_\mathbf{x}, \mathbf{f}'_\mathbf{x}, \mathbf{x}^{(n)}) &= \mathbf{x}^{(n)} \cap (m(\mathbf{x}^{(n)}) - \tfrac{\mathbf{f}_\mathbf{x}(m(\mathbf{x}^{(n)}))}{\mathbf{f}'_\mathbf{x}(\mathbf{x}^{(n)})}) \end{aligned}$$

The function MonoNewton computes the fix-point of $N(\mathbf{f}_\mathbf{x}, \mathbf{f}'_\mathbf{x}, \mathbf{x})$ and returns the resulting interval. The evaluation of the division $\mathbf{f}(m(\mathbf{x}^{(n)}))/\mathbf{f}'(\mathbf{x}^{(n)})$ with extended interval arithmetic [2,17] may produce a gap as illustrated on example 3 below.

Example 3. Let $f(x, y) = x^2 - y$ with $x \in [-4, 4]$ and $y \in [1, 16]$. The interval functions $\mathbf{f}_\mathbf{x}$ and its derivative $\mathbf{f}'_\mathbf{x}$ are defined by $\mathbf{f}_\mathbf{x}(x) = x^2 - [1, 16]$ and $\mathbf{f}'_\mathbf{x}(x) = 2x$. Then,

$$\begin{aligned} \mathbf{x}^{(0)} &= [-4, 4] \\ \mathbf{x}^{(1)} &= [-4, 4] \cap (0 \ominus ((0^2 \ominus [1, 16]) \oslash (2 \otimes [-4, 4]))) \\ &= [-4, 4] \cap ([1, 16] \oslash [-8, 8]) \\ &= [-4, -1/8] \cup [1/8, 4] \end{aligned}$$

Thus $N(\mathbf{f}_\mathbf{x}, \mathbf{f}'_\mathbf{x}, \mathbf{x})$ is not in general a single interval but may be a union of intervals. Let us denote by MonoNewton*, the function that returns this union of intervals. We

LeftNarrow(in:$\mathbf{f}$,$\mathbf{f}'$,$\mathbf{x}$): Interval

1: $r \leftarrow \overline{x}$
2: **if** $0 \notin \mathbf{f}(\mathbf{x})$ **then return** $\emptyset$
3: $\mathbf{x} \leftarrow \mathsf{MonoNewton}(\mathbf{f}, \mathbf{f}', \mathbf{x})$
% 1. MonoNewton may produce a gap
% 2. A gap appears
% when right bound is reduced
4: **if** $0 \in \mathbf{f}([\underline{x}, \underline{x}^+])$ **then**
5: **return** $[\underline{x}, r]$
6: **else**
7: $l \leftarrow \mathsf{LeftNarrow}(\mathbf{f}, \mathbf{f}', [\underline{x}, m(\mathbf{x})])$
8: **if** $l = \emptyset$ **then**
9: $l \leftarrow \mathsf{LeftNarrow}(\mathbf{f}, \mathbf{f}', [m(\mathbf{x}), \overline{x}])$
10: **endif**
11: **return** $[l, r]$
12: **endif**
13: **return** $\mathbf{x}$

(a) LeftNarrow

LeftNarrow*(in:$\mathbf{f}$,$\mathbf{f}'$,$\mathbf{x}$, in-out: S, $\mathbf{u}$): Interval

1: $r \leftarrow \overline{x}$
2: **if** $0 \notin \mathbf{f}(\mathbf{x})$ **then return** $\emptyset$
3: $\mathbf{u}' \leftarrow \mathsf{MonoNewton}^\star(\mathbf{f}, \mathbf{f}', \mathbf{x})$
% Mind the gap
4: $\mathbf{u} \leftarrow \mathbf{u} \cap_{\mathbb{U}} (\mathbf{u}' \cup [r, +\infty))$
5: $\mathbf{x} \leftarrow \Box_{\mathbb{I}}(\mathbf{u})$
6: **if** $0 \in \mathbf{f}([\underline{x}, \underline{x}^+])$ **then**
7: **return** $[\underline{x}, r]$
8: **else**
9: $l \leftarrow \mathsf{LeftNarrow}^\star(\mathbf{f}, \mathbf{f}', [\underline{x}, m(\mathbf{x})], S)$
10: **if** $l = \emptyset$ **then**
11: $l \leftarrow \mathsf{LeftNarrow}^\star(\mathbf{f}, \mathbf{f}', [m(\mathbf{x}), \overline{x}], S)$
14: **endif**
15: **return** $[l, r]$
16: **endif**
17: **return** $\mathbf{x}$

(b) LeftNarrow*

Fig. 7. Box narrowing operators and its variant which collect the gaps

define the LeftNarrow* (see figure 7(b)), which extends the classical box narrowing operator to collect the gaps. MonoNewton* collects the gaps produced by the extended interval newton method (line 3). The gap produced by the right bound contraction is collected in line 4.

5 Experimental Results

This section reports experimental results of MindTheGaps on a variety of classical benchmarks : two classical benches of interval arithmetics (*i1, i4*), an application of robot kinematics (*kin1*), some applications of economic modeling (*eco7* up to *eco10*), some problem made of Euclidean distance equations (*ponts*, *ext-penta* and some particular instances) and a polynomial system from the Posso test suite (*caprasse*). More details on *i1, i4, kin1* and *ecoN* can be found in [7], *ponts* in [21], *ext-penta* in [22] and *caprasse* in [23].

5.1 Customizing MindTheGaps

This section introduces three categories of strategies for customizing MindTheGaps. These heuristics have been investigated to answer the three following questions :

1. Gap validation : Which gaps are not relevant and should not be considered ?
2. Domain selection : Among the domains for which gaps have been found, which ones are the more relevant to split ?
3. Gap splitting : Given one or more selected domains, how to perform the splitting ?

To answer these questions, the following strategies have been explored :

- *Gap validation strategies* : Suppose that a gap has been identified by the filtering algorithms in the domain $\mathbf{x} = [a, d]$, such that $\mathbf{u} = [a, b] \cup [c, d]$. Two different strategies have been explored to validate the gap (b, c), depending on its position within the domain or its relative size :
  - Hansen [2] Keep (b, c) if $\min\{d - b, c - a\} \geq 0.25w(\mathbf{x})$. This strategy eliminates gaps strictly included in one of the extremal quarters of the domain.
  - Large-Gaps : Keep (b, c) if $c - b \geq 0.1w(\mathbf{x})$. This strategy eliminates small gaps with respect to the width of the domain.

  Note that these two strategies can be combined. By default, all the gaps identified by the filtering algorithms are kept (AllGaps).
- *Domain selection strategies* : Different heuristics have been explored :
  - LW (Largest Width) / SW (Smallest Width) : the selected domain holds the largest (resp. smallest) gap found [2].
  - LRW (Largest Relative Width) / SRW (Smallest Relative Width) : the selected domain maximizes (resp. minimizes) the ratio between gap width and domain width.
  - LTW (Largest Total Width) / STW (Smallest Total Width) : The selected domain maximizes (resp. minimizes) gap width sum.
- *Gap splitting strategies* :As mentioned above, three different strategies for splitting have been explored :
  - B1G (Bisect One Gap) : use only one gap to split the selected domain, and generate 2 subproblems [12].
  - BkG (Bisect k Gaps) : use k gaps in the selected domain to split the box and generate $k + 1$ subproblems [1].
  - M3G (Multisect 3 Gaps) : use at most three gaps in three different domains, and combine the subdomains to generate up to 8 subproblems [2]. The three domains and the gaps are determined by the domain selection strategy.

5.2 Analysis of Experimental Results

The above mentioned different strategies have been experimented on various benchmarks. However, due to lack of space, the tables presented in this section are limited to the results obtained with RR, LW and B1G. The other strategies did provide very similar results for most benches, except for the one explicitly discussed in the rest of the section.

All the tests use Realpaver [15] version 0.3. The tests have been run on a Pentium IV at 2.6Ghz running Linux. MindTheGaps has been implemented on the top of Realpaver. The different strategies, as well as the gap gathering process, have been added to the default Realpaver algorithm[4]. Note however that the Box filtering algorithm has been modified in order to fit to the default algorithm which made use of a univariate Newton algorithm[5].

[4] Note that the multivariate Newton algorithm has also been extended to collect gaps [2].

[5] The Box implementation of Realpaver does not use a univariate Newton algorithm. It only relies on interval computation to exclude some subparts of the domain.

Table 1. Experimental results for HC4 on the left and HC4+Newton on the right. The results were obtained using RR, LW and B1G.

	Filtering: HC4							Filtering: HC4+Newton						
	RR		MindTheGaps(RR)			Ratio		RR		MindTheGaps(RR)			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	t(s)	B	H	t	B
eco7	57.07	754885	14.74	231595	1	**-74%**	**-69%**	61.99	468799	12.74	107817	11	**-79%**	**-77%**
eco8	133.51	1614495	112.77	1360061	1	**-15%**	**-16%**	56.77	353155	40.54	246927	49	**-29%**	**-30%**
ponts	34.19	174915	33.12	171251	1043	**-3%**	**-2**	25.61	32643	16.80	21025	946	**-35%**	**-36%**
ponts0	0.30	1395	0.29	1395	0	-	-	0.06	71	0.07	71	0	-	-
ponts1	5.19	26523	4.54	22475	274	**-13%**	**-16%**	5.78	7465	3.61	4563	254	**-38%**	**-39%**
ponts2	23.63	123585	22.93	120993	779	**-3%**	**-2%**	18.77	24009	12.33	15567	670	**-35%**	**-35%**
pentagon	0.60	6131	0.59	5891	52	**-2%**	**-4%**	0.26	1655	0.23	1415	52	**-12%**	**-15%**
ext-penta	-	-	-	-	-	-	-	474.92	1006031	437.37	890943	11723	**-8%**	**-12%**
ext-penta0	0.34	873	0.11	423	51	**-68%**	**-52%**	0.45	873	0.17	423	51	**-62%**	**-52%**
ext-penta1	0.32	263	0.05	255	17	**-85%**	**-3%**	0.39	263	0.10	255	17	**-74%**	**-3%**
ext-penta2	0.77	2825	0.25	2047	9	**-68%**	**-28%**	1.23	2825	0.60	2047	9	**-51%**	**-28%**
i1	29.60	515909	28.75	501677	82	**-3%**	**-3%**	49.46	340057	53.73	370449	38	8%	9%
i4	0.83	2047	0.77	2047	1023	**-8%**	-	1.19	2047	1.07	2047	1023	**-10%**	-
kin1	25.69	264685	19.99	203987	1	**-22%**	**-23%**	0.41	1447	0.30	1263	1	**-27%**	**-13%**
caprasse	0.79	6527	0.80	6527	0	1 %	-	0.65	2567	0.65	2567	0	-	- %

In tables $1--4$, t is the execution time in seconds ("-" signifies more than 1 hour), B is the total number of boxes generated by the splitting process and H is the number of splits in a gap. The column ratio introduces the reduction percent in terms of CPU time (t) and total number of branchings (B).

Table 1 displays the results for a search combined with a HC4 filtering and a search combined with a HC4 interleaved with a multivariate newton algorithm. In both cases, MindTheGaps improves significantly the execution time and reduces the number of splitting. For example, on *eco7*, the execution time is reduced by a factor of 3.8 or more, depending on the kind of consistency we use, and the number of splits is reduced by a factor of 3 up to 4. Even on problems where the number of splits is left unchanged, like on *i4*, the MindTheGaps strategy has still room to improve the execution time. This example underline the key role of the cutting direction in the search process.

Other strategies than choosing the variable with the biggest gap and splitting on this gap do not change significantly the results. However, when applied to a Box filtering or a Box filtering interleaved with a multivariate Newton (see table 3), the strategies which rely on selecting the gaps lying at the center of the domain (Hansen's strategy) or which reject the smallest gaps improve the search on some problems (see table 2). For example, it solves *eco8* in less than half an hour while other strategies require more than an hour. This success is largely due to the way Box filtering produces gaps. Box filtering attempts to reject some part of the variable domains lying at the bounds. As a result, it tends to produce more gaps and smaller gaps near the bounds of the domains. This behavior is exemplified on *ext-penta2*, where the number of gaps used by the search goes from 7037 gaps down to 9 gaps (see table 2). The same remarks can be done when the Box filtering is combined with a multivariate Newton though this last smooth the effect of these strategies. However, whatever the strategies, MindTheGaps still succeeds in improving the execution time over a classical round robin.

MindTheGaps offers others advantages than improving search performances. For example, on the well known *combustion* benchmark, MindTheGaps provides the four

Table 2. Experimental results for Box on the left and on the right, Box with Hansen's criterion for eliminating small gaps. The results were obtained using RR, LW and B1G.

	Filtering: Box											
	RR		MindTheGaps(RR)			Ratio		MindTheGaps(RR)+Hansen			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	H	t	B
eco7	995.12	595505	-	-	-	-	-	276.91	192699	104	**-72%**	**-68%**
eco8	-	-	-	-	-	-		1372.13	847373	177	-	-
ponts	659.70	173331	644.60	170721	968	**-2%**	**-2%**	676.36	172515	1015	3%	**-1%**
ponts0	5.60	1481	4.90	1203	6	**-12%**	**-10%**	5.61	1481	0	-	- %
ponts1	105.47	25943	103.67	23335	122	**-2%**	**-10%**	99.83	22911	255	**-5%**	**-12%**
ponts2	461.20	122865	440.99	118123	656	**-5%**	**-4%**	459.61	121017	538	**-1%**	**-2%**
pentagon	11.27	6283	11.02	6275	173	**-3%**	- %	10.99	6033	51	**-3%**	**-4%**
ext-penta	-	-	-	-	-	-	-	-	-	-	-	-
ext-penta0	5.80	873	2.20	1771	666	**-62%**	102.86%	1.84	423	51	**-68%**	**-52%**
ext-penta1	6.65	263	1.09	873	306	**-84%**	232%	0.81	255	17	**-88%**	**-3%**
ext-penta2	11.36	2825	12.05	17865	7037	6%	533%	2.12	2047	9	**-82%**	**-28%**
i1	353.70	484511	369.60	502035	7614	5%	4%	353.67	482565	39	-	-
i4	5.51	2047	6.07	2047	1023	10%	-	6.05	2047	1023	10%	-
kin1	235.74	132547	-	-	-	-	- %	217.18	120309	56	**-8%**	**-10%**
caprasse	10.55	2023	10.50	1991	48	- %	**-2%**	10.52	1991	48	- %	**-2%**

Table 3. Experimental results for Box+Newton on the left and on the right, Box+Newton with Hansen's criterion for eliminating small gaps. The results were obtained using RR, LW and B1G.

	Filtering: Box+Newton											
	RR		MindTheGaps(RR)			Ratio		MindTheGaps(RR)+Hansen			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	H	t	B
eco7	797.72	429263	207.32	109267	685	**-74%**	**-75%**	196.39	102487	145	**-76%**	**-76**
eco8	-	-	-	-	-	-	-	516.92	224513	371	-	-
ponts	163.31	31735	180.84	35475	1098	11%	12%	133.51	25829	1014	**-19%**	**-19%**
ponts0	0.47	71	0.47	71	0	-	-	0.47	71	0	-	- %
ponts1	33.78	7363	27.56	4981	261	**-18%**	**-32%**	27.39	4981	261	**-19%**	**-32%**
ponts2	117.27	23417	110.37	21881	646	**-6%**	**-7%**	96.50	18675	563	**-18%**	**-21%**
pentagon	3.76	1639	0.27	1399	62	**-93%**	**-15%**	3.45	1399	62	**-9%**	**-15%**
ext-penta	-	-	-	-	-	-	-	-	-	-	-	-
ext-penta0	6.43	873	1.58	707	133	**-75%**	**-19%**	2.02	423	51	**-69%**	**-52%**
ext-penta1	7.18	263	1.12	747	209	**-84.40%**	184%	0.92	303	41	**-88%**	15%
ext-penta2	12.55	2825	7.28	8721	3167	**-42%**	208.70 %	2.57	2047	9	**-80%**	**-28** %
i1	247.22	309681	244.86	302397	576	**-1%**	**-3%**	247.38	305797	52	-	**-1%**
i4	6.26	2047	6.84	2047	1023	9%	-	6.82	2047	1023	9%	-
kin1	2.78	791	1.87	641	72	**-33%**	**-19%**	1.86	629	66	**-33%**	**-20%**
caprasse	10.54	1495	10.52	1463	48	-	**-2%**	10.54	1463	48	-	**-2%**

solutions when a basic round robin found only two enclosures of the four solutions[6]. Here, MindTheGaps takes benefit of two gaps found by HC4 or a Box to isolate the four solutions.

5.3 Gaps and Constraint Evaluation

Factorization rules have been designed for univariate or multivariate polynomials [24]. These symbolic tools aim at reducing the negative effects of interval computations. In general, the evaluation of polynomial constraints in factorized form is tighter. Similarly,

[6] These results were obtained with the default precision of Realpaver (1.0e-8). When the precision is increased, then Realpaver find all the solutions with a basic search strategy.

Table 4. Experimental results for *ecoN* and its corresponding Horner form *ecoNH*

	Filtering: HC4+Newton						
	RR		MindTheGaps(RR)			Ratio	
	t(s)	B	t(s)	B	H	t	B
eco6	1.04	12087	0.56	6383	3	**-46.15%**	**-47.19%**
eco6H	0.69	9301	**0.29**	3729	1	**-57.97%**	**-59.90%**
eco7	61.99	468799	**12.74**	107817	11	**-79.44%**	**-77.00%**
eco7H	49.78	412957	8.42	82143	4	**-83%**	**-80%**
eco8	56.77	353155	40.43	246927	49	**-28.78%**	**-30.07%**
eco8H	30.93	216955	**24.17**	164733	4	**-21.85%**	**-24.07%**
eco9	636.75	2931479	641.75	2934801	1720	.78%	.11%
eco9H	301.20	1541855	**233.67**	1303655	103	**-22.42%**	**-15.44%**
eco10	7569.05	25751025	7381.65	24939453	17949	**-2.47%**	**-3.15%**
eco10H	554.72	2620443	**475.00**	2156345	808	**-14.37%**	**-17.71%**

it should provide tighter gaps within the domains. Moreover, it may provide gaps that might have not been identified using the developed form :

Example 4. Let $c : x^2 + x * y = 1/2$ and its factorized form $c' : x(x + y) = 1/2$, with $\mathbf{x} = \mathbf{y} = [-1, 1]$. $\Box_{\mathbb{U}}(\pi^x_c(\mathbf{X})) = [-1, 1]$ while $\Box_{\mathbb{U}}(\pi^x_{c'}(\mathbf{X})) = [-1, 0.25] \cup [0.25, 1]$.

We have performed some experimentations on *ecoN* to compare the developed form and the corresponding Horner form (see table 4). These experimentations clearly show that Horner form provides a significant improvement (factor 2 to 15) with respect to the classical form for standard bisection. The number of Gap splitting (H) performed by MindTheGaps is strongly reduced (for instance by a factor 16 for *eco9*). However, the impact of MindTheGaps both on computation time and number of branching is stronger.

6 Extension to Partial Consistencies

kB-consistencies are not strictly local consistencies. Informally speaking, these higher consistencies try to shrink the domain by proving that no solution exists in some part of the domain. To do so, they use a lower order consistency ($(k-1)$B-consistency). The point is that they only reduce the bounds of the domains. kB-consistency has a recursive definition based on 2B-consistency which is equivalent to Hull-consistency. Bound-consistency [10] is similar to 3B-consistency but it is based on Box-consistency, rather than 2B-consistency. Thus, partial consistencies also allow to identify gaps within the domains. Whenever kB-consistency tries to refute some interval $\alpha \subset \mathbf{x}_i$, it applies a $(k-1)$B-consistency over $P_{\mathbf{x}_i \leftarrow \alpha}$. Suppose that α is not eliminated but reduced to α'. Then, gaps can be retrieved in three different ways :

1. $\alpha \setminus \alpha'$ is a gap for $\mathbf{x}_i$
2. The gaps found by $(k-1)$B-consistency within α' holds also for $\mathbf{x}_i$.
3. The gaps found during the filtering of $P_{\mathbf{x}_i \leftarrow \alpha}$ within the domains of the other variables are only valid if they have been found also during the filtering of $P_{\mathbf{x}_i \leftarrow \mathbf{x}_i \setminus \alpha}$

For example, let $\mathbf{C} = \{x^2 + y^2 = 1, y = -x^2 + 1\}$, with $\mathbf{x} = \mathbf{y} = [-10, 10]$. A 2B-consistency filtering reduces $\mathbf{x}$ to $[-0.99, 0.99]$ and $\mathbf{y}$ to $[0.1, 1]$. Then, consider that $\mathbf{y}$ is split in two parts $\mathbf{y}_1 = [0.55, 1]$ and $\mathbf{y}_2 = [0.1, 0.55]$:

- $\mathbf{X}_1 = (\mathbf{x}, \mathbf{y}_1)$ is reduced by 2B-consistency to $([-0.54, 0.54], [0.8, 1])$. Thus, there is no solution for $y \in (0.55, 0.8)$ (pruned by 2B-consistency (case 1)). Moreover, a gap has been identified for variable $x : \mathbf{x}_2 = [-0.54, -0.316] \cup [0.316, 0.54]$.
- $\mathbf{X}_2 = (\mathbf{x}, \mathbf{y}_2)$ is not reduced by 2B-consistency, but a gap is identified for variable $x : \mathbf{x}_1 = [-1, -0.84] \cup [0.84, 1]$.

Consequently, if $y \in \mathbf{y}_1$ or $y \in \mathbf{y}_2$, then the set of allowed values for variable x is $\mathbf{x}_1 \cup \mathbf{x}_2$, that is to say $[-1, -0.84] \cup [-0.54, -0.316] \cup [0.316, 0.54] \cup [0.84, 1]$ (case 3).

Note that, the bound reductions (case 1) have been used in [25] to improve kB-consistency complexity. The gaps produced in case 2 and 3 could be used in a similar way to improve kB-consistency efficiency. A very first implementation of MindTheGaps combined with 3B-consistency shows significant improvements.

7 Conclusion

We have introduced in this paper a new splitting strategy for search algorithm in nonlinear CSPs. This splitting strategy takes advantage of the gaps generated by consistency filtering algorithms. These gaps provide indications for selecting which domain to split and for selecting cutting points inside the domains. Splitting the domain by removing such gaps definitely reduces the search space. It also helps to discard some redundant solutions and helps the search algorithm to isolate different solutions. Experimental results show that in numerous problems, the performances of the search process are significantly improved in comparison with classical search algorithm.

References

1. Ratz, D.: Box-splitting strategies for the interval Gauss–Seidel step in a global optimization method. Computing **53** (1994) 337–354
2. Hansen, E.: Global optimization using interval analysis. Marcel Deckler (1992)
3. Kearfott, R.: Rigorous global search: continuous problems. Kluwer (1996)
4. Lhomme, O.: Consistency techniques for numerical csps. In: IJCAI-93. (1993) 232–238
5. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.: Revising hull and box consistency. In: International Conference on Logic Programming. (1999) 230–244
6. Benhamou, F., McAllister, D., Van Hentenryck, P.: CLP(intervals) revisited. In Bruynooghe, M., ed.: International Symposium of Logic Programming. MIT Press (1994) 124–138
7. Van Hentenryck, P., McAllister, D., Kapur, D.: Solving polynomial systems using a branch and prune approach. SIAM, Journal of Numerical Analysis **34(2)** (1997) 797–827
8. Collavizza, H., Delobel, F., Rueher, M.: Comparing partial consistencies. Journal of Reliable Computing **5** (1999) 213–228
9. Lebbah, Y.: Contribution à la résolution de contraintes par consistance forte. Thèse de doctorat, École des Mines de Nantes (1999)
10. Puget, J., Van Hentenryck, P.: A constraint satisfaction approach to a circuit design problem. Journal of Global Optimization **13** (1998) 75–93
11. Jussien, N., Lhomme, O.: Dynamic domain splitting for numeric CSPs. In: European Conference on Artificial Intelligence. (1998) 224–228
12. Hansen, E., Greenberg, R.: An interval newton method. Applied Mathematics and Computations **12** (1983) 89–98

13. Hyvönen, E.: Constraint reasoning based on interval arithmetic: the tolerance propagation approach. Artificial Intelligence **58** (1992) 71–112
14. ILOG: Solver Reference manual http://www.ilog.com/product/jsolver. (2002)
15. Granvilliers, L.: Realpaver: Solving non linear constraints by interval computations. User's manual (2003) http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver.
16. Macworth, A.: Consistency in networks of relations. Artificial Intelligence (1977) 99–118
17. Moore, R.: Interval analysis. Prentice-Hall (1977)
18. Kearfott, R.: A review of techniques in the verified solution of constrained global optimization problems. In Kearfott, R.B., Kreinovich, V., eds.: Applications of Interval Computations. Kluwer, Dordrecht, Netherlands (1996) 23–59
19. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer (2001)
20. Lhomme, O.: Contribution à la résolution de contraintes sur les réels par propagation d'intervalles. Thèse de doctorat, Université de Nice-Sophia Antipolis (1994)
21. Jermann, C., Trombettoni, G., Neveu, B., Rueher, M.: A constraint programming approach for solving rigid geometric systems. In: Proc. of CP'00, Singapore (2000) 233–248
22. Batnini, H., Rueher, M.: Décomposition sémantique pour la résolution de systèmes d'équations de distances. JEDAI **2** (2004) Édition spéciale JNPC 2003.
23. Traverso, C.: The posso test suite examples (2003) http://www.inria.fr/saga/POL/index.html.
24. Ceberio, M.: Contribution à l'étude des CSPs numériques sous et sur-contraints. Outils symboliques et contraintes flexibles continues. PhD thesis, Université de Nantes (2003)
25. Bordeaux, L., Monfroy, E., Benhamou, F.: Improved bounds on the complexity of kb-consistency. In Kaufmann, M., ed.: Proceeding of IJCAI'2001. (2001) 303–308