New Light on Arc-Consistency over Continuous Domains

Gilles Chabert, Gilles Trombettoni and Bertrand Neveu

PROJET COPRIN I3S-INRIA-CERTIS, 2004 route des Lucioles BP 93
06902 Sophia Antipolis Cedex, FRANCE
{chabert, trombe, neveu}@sophia.inria.fr

Abstract. Hyvönen [5] and Faltings [4] observed that propagation algorithms with continuous variables are computationally extremely inefficient when unions of intervals are used to precisely store refinements of domains.
These algorithms were designed in the hope of obtaining the interesting property of arc-consistency, that guarantees every value in domains to be consistent w.r.t. every constraint.
In this paper, we show that a pure backtrack-free filtering algorithm enforcing arc-consistency will never exist. But surprisingly, we show that it is easy to obtain a property stronger than arc-consistency with a few steps of bisection.
We define this so-called *box-set consistency* and detail an efficient algorithm to enforce it.

1 Introduction

Solving systems of nonlinear equations over the reals with interval constraint programming usually resorts to a combination of local filtering, interval analysis and domain splitting.

Local filtering techniques are based on an interval narrowing operator, called *projection*, that computes compatible values for different variables linked by a constraint. Sometimes, the projection results in a union of intervals: for instance with the constraint $x^2 = y$, if y varies within $[1, 4]$ then the domain of variation for x obtained by projection is either $[-2, -1]$ or $[\,1,\,2]$.

In this case, the *hull consistency* algorithm [1] (also known as *2B consistency* [6]) computes the enclosing interval of the union immediately, therefore losing track of each "gap" between intervals. Another approach, inspired by the well-known *arc-consistency*, would be to store and propagate unions of intervals.

Arc-consistency has never been applied successfully. We explain with an example that this failure is not due to bad algorithmic choices, but to a property inherent to continuous CSPs.

Yet, we show that it is possible in a solving strategy that includes a specific splitting technique, called *natural splitting*, to obtain boxes that verify not only arc consistency but another property called *box-continuity*. The lazy version of this new algorithm causes no time overhead in practice and gives promises.

The paper is organized as follows. In section 2, we sum up the concepts of constraint programming over the reals. In section 3, we explain why arc-consistency cannot be achieved. We define a stronger consistency and show related properties. In section 4, an algorithm that enforces this consistency is given.

2 Background

2.1 Constraint Reasoning over the Reals

A **numerical constraint satisfaction problem** (NCSP) is a 3-uple (C,V,B). C is a set of constraints $c_1, ..., c_m$ (equations or inequations) relating a set V of variables $x_1, ..., x_n$. Each variable is given an initial domain of real values $D_{x_1}, .., D_{x_n}$, and the problem is to find all the n-tuples of values $(v_1, ..., v_n)$, $v_i \in D_{x_i}$ $(1 \leq i \leq n)$, such that constraints are all satisfied when simultaneously each variable x_i is assigned to v_i. Such a n-tuple is called a *solution*. Usually, domains are represented by intervals and a *box* designates a cartesian product of domains. $B = D_{x_1} \times ... \times D_{x_n}$ is the initial box of the problem. In this paper, we will resort also to a more complex representation of domains, where a variable domain is assigned a union of intervals. In this case, the cartesian product B of domains will be called a *h-box* (a box with "gaps").

We will use intensively a relation of problem inclusion. Let us define it once and for all.

Definition 1 (Sub-NCSP). *Let $P = (C, V, B)$ and $P' = (C', V', B')$ be two NCSP. P is included in P' iff $C = C'$, $V = V'$ and $B \subset B'$*

In practice, NCSP are large nonlinear problems that are intractable by symbolic solving techniques. Traditional numerical methods do not suit either because we are looking for all the solutions. Solving can be achieved by combining local filtering, domain splitting, and interval analysis.

Local filtering techniques refine domains of variables thanks to partial properties of the problem, that is, properties which hold on subproblems. These techniques converge in polynomial time, and the resulting box (or h-box) is said to be *locally* consistent.

The general scheme for finding solutions consists in a search tree, where local filtering is enforced at each node. Once local consistency is reached, the domain of one variable is chosen and split in two sub-domains, which leads to two subnodes in the tree.

A major approach for local filtering is the well-known *hull consistency*, obtained by a Waltz-like propagation algorithm [9]. The algorithm is detailed further. Here are the underlying concepts :

- **Projection**: Refine the domain of a variable x with respect to a specific constraint c, using interval arithmetics [8].
- **Propagation**: Propagate reductions over the other variables linked to x by another constraint c'.

2.2 Projections

Let c be a binary constraint relating variables x and y. We denote Π_x^c the *projection* [1] *of c over x*, a function that takes an h-box $B = D_{x_1} \times ... \times D_x \times ... \times D_y \times ... \times D_{x_n}$ as input and computes all possible values for x in D_x as y varies within D_y. Formally, if D'_x is the result of Π_x^c applied on B, we have: $D'_x = \{v \in D_x \mid \exists w \in D_y, \ \ c(v,w) \text{ is satisfied}\}$. This definition can be easily generalized to k-ary constraints:

Definition 2 (Projection). *Let c be a constraint relating variables $x, y_1, ..., y_k$. We call projection of c over x the following function:*
$\Pi_x^c : B \rightarrow \{v \in D_x \mid \exists (v_1, ..., v_k) \in D_{y_1} \times ... \times D_{y_k}, \ \ c(v, v_1, ..., v_k) \text{ is satisfied}\}$

Example 1. $c : x + y = z$
$\Pi_x^c : D_x \times D_y \times D_z \longrightarrow (D_z \ominus D_y) \cap D_x$
where $\ominus$ is the natural extension of the arithmetic operator minus. Note that computing this projection requires also an intersection with D_x.

Basically, the projection of a constraint $f(y, x_1, ..., x_n) = 0$ over y requires to find an implicit function ϕ such that $f(y, x_1, ..., x_n) = 0 \Leftrightarrow y = \phi(x_1, ..., x_n)$.

Sometimes, there is not a unique implicit function but several continuous functions $(\phi_1, \phi_2, ...)$, then we talk about *disjunction* and in this case Π_y^c gives a union of intervals:

Example 2. $c : x^2 = y$
Π_x^c applied on $D_x \times D_y$ with $D_x = [-2, 2]$ and $D_y = [1, 4]$ gives the union $D'_x = [-2, -1] \cup [\,1,\ 2]$

The set of values returned by a projection may be either an interval (example 1) or a union of intervals (example 2). To place our discussion in the most general case, we consider hereinafter that a projection returns a set U of disjoint intervals, that we will call abusively a *union*, and write $|U|$ the number of intervals contained in U [2].

2.3 Propagation

Modifying (or revising) the domain of a variable may have repercussions on the other variables. *Propagate* means to memorize in a queue (or an agenda) all the pairs $< c, x >$ of constraint/variable such that the projection of c over x can be effective. When the queue is empty, we are sure that no more reduction is possible and that we have reached a fix point.

[1] Also called *Solution function* [5].

[2] Unions of disjoint intervals could be defined algebraically with their operators and their arithmetic. But their use is rather intuitive, so we will not give such a formalism here. Sometimes we just substitute unions for intervals, to avoid to overwhelm this paper with definitions.

In this paper, we will refer to a procedure `Propagate(NCSP` (C, V, B)`, in-out Queue` Q`, Constraint` c`, Variable` x`)`, that updates the propagation queue Q after a projection of c over x.

If revising a domain consists in applying the projection operator Π defined above, the resulting NCSP is *arc-consistent*:

Definition 3 (Arc-Consistency). *Let* $P = (C, V, B = D_{x_1} \times ... \times D_{x_n})$ *be a NCSP.*
P is arc consistent iff $\forall < c, x > \in C \times V$ *with* x *related by* c, $D_x = \Pi_x^c(B)$

Remark 1. In this paper, we will talk about the *arc-consistency of a box (or an h-box)* B to designate the arc-consistency of the problem (C, V, B) with the set C and V given by the context.

We will see in section 3.1 that arc-consistency is not feasible with continuous variables, so usually the revising operation is not the projection itself, but an outer approximation. Computations are all interval-based and this operator avoids to manage unions. The resulting problem is *hull-consistent*:

Definition 4 (Hull-Consistency). *Let* $P = (C, V, B = D_{x_1} \times ... \times D_{x_n})$ *be a NCSP.*
P is hull consistent iff $\forall < c, x > \in C \times V$ *with* x *related by* c, $D_x = \Box\Pi_x^c(B)$

The symbol $\Box$ stands for the *hull* operation. Example: $\Box\{[0,1],[2,3]\} = [0,3]$.

Propagation, arc-consistency, and hull consistency are extensively covered in literature, see [2] for example. To summarize, here is a generic algorithm `HC_Filtering` of hull consistency filtering.

Procedure 1 `HC_Filtering`(NCSP (C, V, B))

```
var Q : Queue
for all pairs < c, x > in C × V do
  if x is related by c then
    add < c, x > in Q
while Q is not empty do
  pop a pair < c, x > from Q
  D'_x ⟵ □ Π^c_x(B)
  if D'_x ⊂ D_x then
    Propagate((C, V, B), Q, c, x)
    D_x ← D'_x                    // D_x is the domain of x in B
```

This algorithm originates from Waltz [9] and was applied first over finite domain constraints under the acronym `AC3` [7] to obtain arc-consistency. With intervals, `HC3` [3] introduces a decomposition of the system into *primitive constraints* [3] for which projections can be computed, and `HC4` [2] is an upgraded

[3] A *primitive constraint* is a basic mathematical relation (such as $z = x + y$ or $y = cos(x)$) for which projections are known. A system of standard equations can always be decomposed into an equivalent system of primitive constraints.

version of HC3 that produces the same result sparing decomposition. Both enforce hull-consistency.

3 A stronger property than arc-consistency

In this section, we introduce a new kind of consistency, on a theoretical point of view. The rest of the paper will be devoted to the way it can be enforced. Contrary to arc-consistency, it can be obtained in reasonable time and even more, in some cases, provide improvements compared to the classical approach just given above.

3.1 Arc-Consistency

First of all, let us rule out an ambiguity. Talking about the *arc-consistency of a problem* may have two different meanings, depending on the context. We may refer to the property, which can be either true or false. But we may talk also about the *largest arc-consistent subproblem*. In the latter case, we will use the following definition:

Definition 5 (AC Part). *The AC part of a NCSP is the maximal arc-consistent sub-NCSP, for the order relation of inclusion (see definition 1).*

Example 3. Let $P = (\{x = y\}, \{x, y\}, D_x \times D_y)$ be a NCSP with $D_x = [-1, 1]$ and $D_y = [0, 2]$

In the AC part of P, domains become $[0, 1] \times [0, 1]$. Indeed, any arc-consistent sub-NCSP of P has an h-box with intervals $[\alpha, \beta]$, $0 \leq \alpha \leq \beta \leq 1$, and the (unique) maximal element of these h-boxes is $[0, 1] \times [0, 1]$.

We show below that even with very simple constraints, the AC part of a problem may have a non-representable domain, as an infinity of intervals. Hence, arc-consistency filtering is not applicable over continuous domains, whatever the underlying algorithm is.

We are going to illustrate our claim on the following system of 2 equations:

Example 4. Let $P = (\{c_1, c_2\}, \{x, y\}, B)$ be the following $NCSP$:
$B = D_x \times D_y = [1, 9] \times [1, 9]$
(c_1) : $(\frac{3}{4}(x - 5))^2 = y$
(c_2) : $y = x$

Lemma 1. *In the AC part of P, domains of x and y are an infinity of disjoint non-empty intervals.*

Necessary condition Let f_1 and f_2 be the following (real-valued) functions:
$f_1 : y \longrightarrow \frac{4}{3}\sqrt{y} + 5$
$f_2 : y \longrightarrow 5 - \frac{4}{3}\sqrt{y}$
Let F_1 (resp. F_2) be the "optimal" extensions to intervals [4] of f_1 (resp. f_2), Φ_1 (resp. Φ_2) the extension to unions of intervals associated to F_1 (resp. F_2). Finally, let Φ be the function such that $\Phi(U) = \Phi_1(U) \cup \Phi_2(U)$.

Consider an algorithm that, in turn, computes the following operations: $D_x \leftarrow \Phi(D_y)$ et $D_y \leftarrow D_x$. We omit intersections with domains on purpose, and this is why, *a priori*, we do not call these operations *projections* . Let us denote X_n (resp. Y_n) the domain of x (resp. y) after the n^{th} execution of $D_x \leftarrow \Phi(D_y)$ (resp. $D_y \leftarrow D_x$). X_0 and Y_0 are initial domains.

The figures below depict the first steps of propagation. The h-boxes shown are successively $X_0 \times Y_0$, $X_1 \times Y_0$, $X_1 \times Y_1$ and $X_2 \times Y_2$.

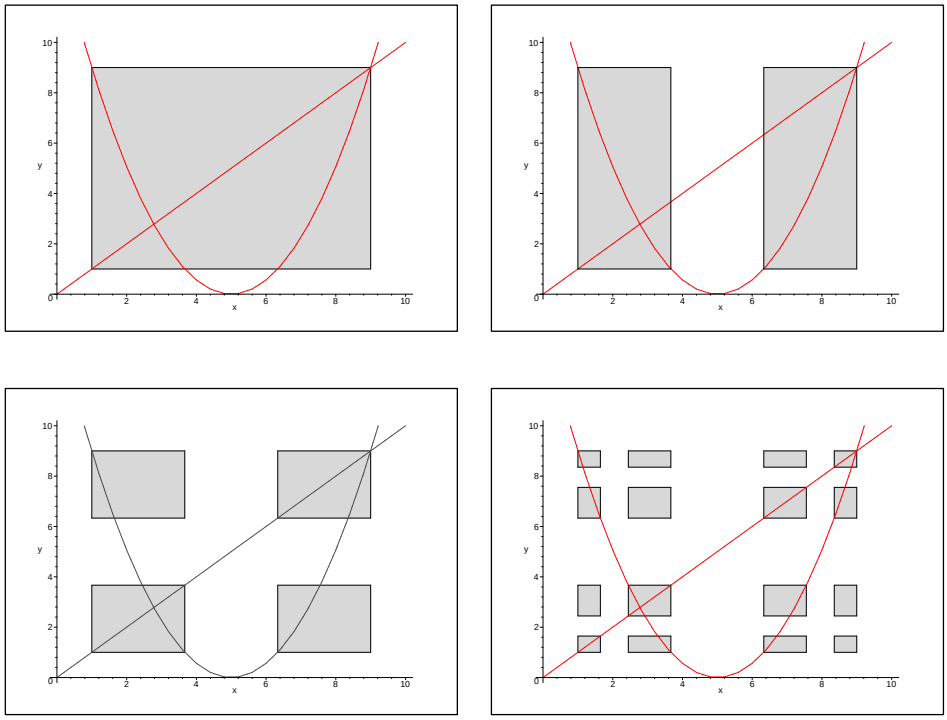


Fig. 1. First steps of `AC` filtering

As we see, the size of unions grows exponentially. Let us show some properties of this algorithm.

[4] F is optimal iff for any interval I, $F(I) = \square f(I)$

Property 1. $\forall n \geq 1,\ \ X_n \subset X_{n-1}$ and $Y_n \subset Y_{n-1}$. In other words, the result of each operation is included in the current domain of the variable.

Proof. By induction. We can check by hand that $X_1 \subset X_0$ and $Y_1 \subset Y_0$. Assume $X_n \subset X_{n-1}$:
$X_n \subset X_{n-1} \Longrightarrow Y_{n+1} \subset Y_n \Longrightarrow \Phi(Y_{n+1}) \subset \Phi(Y_n)$ because interval arithmetic is inclusion monotonic, and then $X_{n+1} \subset X_n$. ▲

Property 2. The number of intervals doubles at each step ($\forall n\ |X_n| = 2 \times |X_{n-1}|$), and more precisely, each interval is split into two disjoint intervals.

Proof. Assume that X_n and Y_n contain p disjoint intervals whose bounds are between 1 and 9. As functions f_1 and f_2 are monotonous on $[1,9]$, $\Phi_1(Y_n)$ and $\Phi_2(Y_n)$ will contain both p disjoint intervals [5]. Still with inclusion monotonicity of interval arithmetic, since $F_1([1,9]) \cap F_2([1,9]) = \emptyset$, the $2 \times p$ intervals obtained will be all disjoint and then $|Y_{n+1}| = |X_{n+1}| = |\Phi(Y_n)| = 2 \times p$.
Moreover, $\Phi(X_0) = \Phi([1,9]) \subset [1,9]$ and thanks to the property 1, we can check that intervals of X_{n+1} and Y_{n+1} are included in $[1,9]$. ▲

Property 3. Bounds of intervals are always maintained in domains. That is to say, if $[a,b]$ is an interval of X_n [6], then $\forall p \geq n$, a and b are interval bounds of X_p.

Proof. First of all, since f_1 and f_2 are monotonous, for any interval I, bounds of $F_1(I)$ and $F_2(I)$ take support on bounds of I. Now, the property is shown by induction: First, bounds 1 and 9 for x and y are always maintained because:
- (x=9, y=9) is a solution of the problem
- x=1 cannot be removed by computing $X \leftarrow \Phi(Y)$ since y=9 is a support.
- y=1 cannot be removed by computing $Y \leftarrow X$ since x=1 is a support.
If we assume now that the bounds of all the intervals in the representation of X_n are maintained for all $p \geq n$, then bounds of X_{n+1} will also be maintained for all $p \geq n+1$ since they take support on bounds of Y_n, i.e. X_n, and they are included in X_n (property 1). ▲

Now, property 1 allows us to say that adding an intersection with domains at each step has no effect. Therefore, this algorithm computes successively projections over x and over y:

$(D_x \leftarrow \Pi_x^{c_1}(B)) \longrightarrow (D_y \leftarrow \Pi_y^{c_2}(B)) \longrightarrow (D_x \leftarrow \Pi_x^{c_1}(B)) \longrightarrow ...$

Property 2 leads immediately to the following fact : The number of intervals in X_n tends to infinity, and even if the size of intervals may tend to zero, each interval contains necessarily one non-removable point (property 3), so we are dealing with an infinity of non-empty disjoint intervals. The AC part of the problem is contained in the result of this algorithm after an infinity of iterations. In a nutshell :

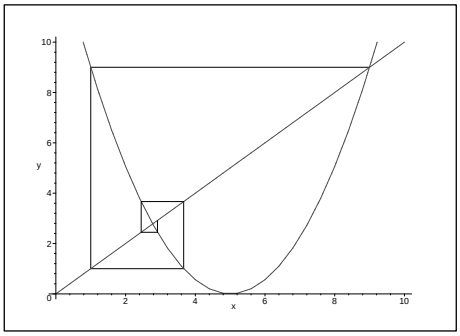
[5] F_1 and F_2 are optimal

[6] We cannot carry on regardless of a bit of rigor here. By saying that $[a,b]$ is an interval of X_n, we mean that $[a,b]$ is an element of an union seen as a set of disjoint intervals. So we consider $[a,b] \in X_n$, and not only $[a,b] \subset X_n$

In the AC part of P, domains are included in an infinity of non-empty disjoint intervals.

Sufficient condition Consider the following numerical series :
$u_0 = 9$
$u_n = f_2(u_{n-1})$

We prove easily that $u_n \to \frac{25}{9}$ when $n \to +\infty$. This convergent series is represented on the following picture :



Let A be the set $\{u_n, n \in \mathbb{N}\} \cup \{\frac{25}{9}\}$. Clearly, the *h-box* $X \times Y = A \times A$ is arc-consistent since each point u_n has a support for both constraints. This *h-box* being included in the initial box $[1,9] \times [1,9]$ of P, then by definition, the AC part of P contains necessarily this *h-box*.

Now, it suffices to observe that for all n, u_n is exactly a bound of an interval of X_n (a bound "discovered" at the n^{th} step of the algorithm above). Proof is similar to property 2: It comes from the monotonicity of f_1 and f_2, and from the fact that $F_1([1,9]) \cap F_2([1,9]) = \emptyset$.

Conclusion We have shown that in the AC part of P, the domain (either for x or for y) includes a set of points u_n (sufficient condition), these points being separated by "gaps" because they are bounds of disjoint intervals (necessary condition). These gaps are inconsistent values that do not belong to the AC part of P. So we have proven the lemma 1.

Remark : In the AC part of P, intervals can be punctual.

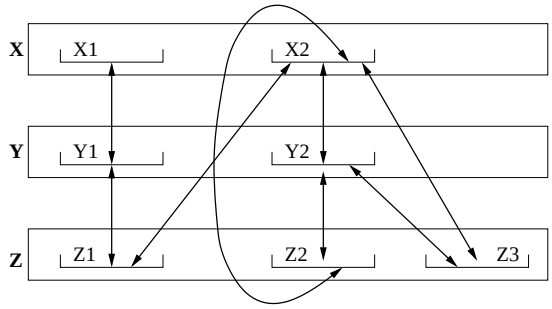
3.2 Box-Set Consistency

We have seen that arc-consistency cannot be achieved over continuous domains. We formally present in this section a stronger consistency that can be achieved.

Let us go back to the example of the previous section, at any step. Let I be an interval of D_x which contains none of the 2 solutions. If we build a box with I and any interval of the current domain D_y, it is not arc-consistent and does not contain any arc-consistent sub-box (see definition 3 and remark 1).

Actually, there are exactly 2 arc-consistent sub-boxes in this example, which are zero-sized boxes around the solutions.

Let us generalize. The following figure depicts a system of 3 variables pairwisely linked by binary constraints. Domains have several intervals, and an arrow between two intervals I and J means that every value of I has a compatible value in J and conversely.



We see that the h-box $(X_1 \cup X_2) \times (Y_1 \cup Y_2) \times (Z_1 \cup Z_2 \cup Z_3)$ is arc-consistent. But there are only two arc-consistent sub-boxes composed with these intervals, which are $X_2 \times Y_2 \times Z_2$ and $X_2 \times Y_2 \times Z_3$.

The box $X_1 \times Y_1 \times Z_1$ is not arc-consistent because X_1 and Z_1 are not linked. Actually, X_1, Y_1 and Z_1 do not belong to any arc-consistent sub-box, and they can be removed from the domains.

In this paper, we present algorithms that find the maximal arc-consistent sub-boxes of a problem.

Definition 6 (Box-set Consistency). *Let $P = (C, V, B)$ be a NCSP. The box-set consistency of P is the set $\{B'\}$ of maximal boxes such that (C, V, B') is an arc-consistent sub-NCSP of P.*

We can either use each of these boxes as a choice point in the original system P (and therefore carry on with splitting), or collect these boxes to get one h-box, which would be $X_2 \times Y_2 \times (Z_2 \cup Z_3)$ in this example.

Box-set consistency is stronger than arc-consistency as the following example illustrates:

Example 5. $D_x = D_y = D_z = [-2, 2]$ $D_w = [1, 4]$
Constraints are $x^2 = w$, $x = y$, $y = z$ and $x = -z$.
Arc-consistency is achieved with the following domains :
$D_x = D_y = D_z = [-2, -1] \cup [1, 2]$, $D_w = [1, 4]$
And box-set consistency discards the whole box (in the domain of x, neither $[-2, -1]$ or $[1, 2]$ belong to an arc-consistent sub-box).

But it is weaker than global solving. It suffices to consider this NCSP:

Example 6. $D_x = [0, 2]$ $D_y = [0, 2]$ $D_z = [0, 2]$
Constraints are $x = y$, $x + z = 2$ and $y = z$.
As the initial box is already arc-consistent, it is box-set consistent. But the real solution is $\{(1, 1, 1)\}$.

3.3 Remark

Box-set consistency can be defined in another fashion. We can characterize arc-consistent boxes as solutions of a problem over the intervals, i.e. a problem where variables take *interval* values instead of *real* values. In this way, Z_1 (in the figure above) would not belong to a solution, and could be discarded as an inconsistent "value". To introduce the definition of this induced problem over the intervals, let us start with a single constraint:

Definition 7 (Arc-Extension of a constraint). *$\mathbb{I}$ represents the set of intervals.*
Let $c(x, y)$ be a constraint relating variables x and y. We call arc-extension of c and denote $arc(c)$ the relation defined on $\mathbb{I} \times \mathbb{I}$ such that:

$$arc(c)(X, Y) \Longleftrightarrow \begin{cases} \forall x \in X,\ \exists y \in Y \mid c(x, y) \\ \forall y \in Y,\ \exists x \in X \mid c(x, y) \end{cases} \quad (1)$$

In a nutshell, $arc(c)(X, Y)$ states that c is arc-consistent. The definition of arc-extension for n-ary constraints is straightforward.

Now the arc-extension of a NCSP is simply defined as follows:

Definition 8 (Arc-Extension of a NCSP). *Let P be a NCSP with m constraints $c_1, ..., c_m$ relating n variables $x_1, ..., x_n$ in an initial box $D_{x_1} \times ... \times D_{x_n}$.*
The arc-extension of P is the set of

- *n variables $X_1, ..., X_n$, with $X_i \in \mathbb{I}$*
- *m constraints $arc(c_1), ..., arc(c_m)$*
- *Initial domains : $\forall i\ (1 \leq i \leq n)\ \ X_i \subset D_{x_i}$*

The solutions of this problem are the maximal arc-consistent boxes.

3.4 Number of boxes of a box-set consistent problem

[5] has shown that the number of intervals for a given variable v in a box-set consistent problem is bounded by $((p-1) \times a) + 1) = O(p \times a)$, where p is the maximum number of intervals obtained by *one* projection, and a is the arity of the variable, that is the number of constraints in which v appears. This leads to a total number of boxes of a box-set consistent problem that is bounded by $(p \times a)^n$. In Hyvönen's terminology, this number bounds the size of the *global application space*.

This result holds on problems with only primitive constraints, and without multiple occurrences of a variable in a same constraint.

The result can easily be extended to problems with any type of constraints by considering, instead of n, the number n' of variables in the decomposed system.

On the contrary, the result seems difficult to hold in the general case. Indeed, the box-set consistency of a problem where the variables with multiple occurrences are renamed does not imply the box-set consistency of the (initial) problem, and we have not found straightforward bounds.

4 The natural splitting algorithm

In this section, we show an algorithm that enforces box-set consistency. This algorithm is based on a strategy of bisection called *natural splitting*. Two versions are presented. Both resort to a projection operator that computes unions, but the second version uses this operator only once per box whereas in the first version it is embedded in a propagation loop.

4.1 The key idea

Let us go back to the algorithm of hull-consistency filtering `HC_Filtering` (see 2.3), and assume that this procedure has been applied on a box B. If for every variable the latest projection has produced only one interval, we will show that B is arc-consistent. If one of the last projections produced at least 2 intervals, the idea is to split the domain of this variable into these 2 intervals. This bisection is called *natural splitting*, to contrast with the semantic-less midpoint splitting.

We hope in this way that such a disjunction will not occur anymore on both sub-boxes. We apply the same process on each of the sub-boxes : hull filtering and natural splitting, until we obtain a fix point.

To distinguish constraints whose projections produce 1 interval from those that produce several intervals, we use the term *box-continuity*. We begin by introducing this notion and give the algorithm afterwards.

4.2 Box-Continuity

Box-continuity is the key property of our approach. We will say that a constraint c is *box-continuous* on a given box when projections of c do not create gaps, i.e. when the result set of a projection of c over whatever variable contains a single interval. Formally:

Definition 9 (Box-Continuity). *Let c be a constraint relating variables $x_1, ..., x_k$, B a box.*
c is box-continuous on $B \iff \forall i \ (1 \leq i \leq k) \ |\Pi^c_{x_i}(B)| = 1$.

Box-continuity is not related to the "classical" mathematical definition of continuity for the functions involved in the constraint : for instance, the function $f_1 : (x, y) \longrightarrow x^2 - y$ is continuous on $B = D_x \times D_y = [-2, 2] \times [1, 4]$ whereas $c_1 : f_1(x, y) = 0$ is not box-continuous since $\Pi_x(c_1)(B) = \{[-2, -1], [1, 2]\}$.

Conversely, $f_2 : (x, y) \longrightarrow I(x - y)$, where $I(z)$ is the integer part of z, is not continuous on $D = D_x \times D_y = [-2, 2] \times [-2, 2]$ whereas $c_2 : f_2(x, y) = 0$ is box-continuous since $\Pi_x(c_2)(B) = \Pi_y(c_2)(B) = \{[-2, 2]\}$

4.3 First version

To perform natural splitting, we need to know where are the gaps produced by the last projections of `HC_Filtering`. One way to retrieve this information immediately is to modify `HC_Filtering` to allow union labeling. When the domain

of a variable is revised, instead of computing the hull of the projection immediately, we can keep a union and computes the hull only when this domain is used as a parameter of another projection. Without changing anything to the algorithm, this trick provides a way to detect box-continuity very easily. Indeed, once hull-consistency is achieved, if the domain of every variable is a single interval, it means that the latest projection performed over any variable resulted in a unique interval, i.e. that all the constraints are box-continuous (on the box).

The following procedure is the first version of our algorithm. It applies the "union" variant of `HC_Filtering`, and split the box as long as a domain in the box contains a gap:

Procedure 2 `Naive_BoxSet`(NCSP (C, V, B), in-out `solutions`)

2: **for all** pairs $< c, x >$ in $C \times V$ **do**
  if x is related by c **then**
4: add $< c, x >$ in Q
  while Q is not empty **do**
6: pop a pair $< c, x >$ from Q
  $D'_x \longleftarrow \Pi^c_x(\Box D_{x_1} \times ... \times \Box D_{x_n})$
8: **if** $(\Box D'_x \subset \Box D_x)$ **then**
    Propagate((C, V, B), Q, c, x)
10: $D_x \leftarrow D'_x$

12: **if** (exists a variable x_i with $|D_{x_i}| > 1$) **then**
  for $j = 1$ to $|D_{x_i}|$ **do**
14: $B \leftarrow \Box D_{x_1} \times ... \times \Box D_{x_{i-1}} \times D^j_{x_i} \times \Box D_{x_{i+1}} \times ... \Box D_{x_n}$
    `Naive_BoxSet`((C, V, B), `solutions`)
16: **else**
  if (B is not empty) **then**
18: add B to `solutions`

Lines 2-10 are the union variant of `HC_Filtering`. Lines 12-15 perform natural splitting.

4.4 Properties

Consider, in the execution of `Naive_BoxSet`, the point where the box B is added to solutions, i.e. at line 18. If we have reached this point, it means that no gap could be found in B, or in other words, that every constraint is box-continuous on B. But B is also hull-consistent so the following proposition applies to B:

Proposition 1. *If every constraint is box-continuous on a box B then:*
B is hull-consistent $\Longleftrightarrow$ B is arc-consistent

Proof. Once the fix point of a hull consistency filtering is reached, we have for every pair $< c, x >$ of constraint/variable: $D_x = \Box\Pi^c_x(B)$. As c is box-

continuous, $\Box\Pi_x^c(B) = \Pi_x^c(B)$ and then $D_x = \Pi_x^c(B)$, which means that the domain of x is arc-consistent regarding c. The converse relation is obvious. ▲

Hence, B is an arc-consistent box. As a rule of thumb: *Hull consistency and Natural Splitting gives the box-set consistency of the problem.*

4.5 Lazy version

In practice, managing unions in `HC_Filtering` is highly inefficient. Moreover, results of projections are computed all along the propagation loop although we are interested only by the last ones. Imagine now that we got a way to check quickly whether a constraint is box-continuous or not. We could apply `HC_Filtering` as it is (without unions), and once the fix point is reached check the constraints one after the other until we find a constraint c that is not box-continuous. If one is found, there is at least one variable x involved in c for which we can exhibit a gap inside the domain. So we compute an exact projection this time to disclose the gap, and finally use it as a candidate for splitting.

So, our solving strategy now is simply a combination of three steps: hull-consistency filtering, gap search, and natural splitting:

Procedure 3 `Lazy_BoxSet`(NCSP (C, V, B), in-out `solutions`)

```
2:  HC_Filtering((C, V, B))

4:  if B is empty then
      return
6:  C' ← C
    found ← false
8:  while (not found) and (C' ≠ ∅) do
      pop c from C'
10:   if c is not box-continuous then
        V' ← the set of variables in V related by c
12:     while (not found) and (V' ≠ ∅) do
          pop x from V'
14:       U ⟵ Π_x^c(B)
          if |U| > 1 then
16:         found ← true

18: if found then
      for all intervals I in U do
20:     D_x ⟵ I
        Lazy_BoxSet((C, V, B), solutions)
22: else
      add B to solutions
```

Line 2 in `Lazy_BoxSet` enforces a hull consistency filtering. In lines 6 to 16, we try to find a gap in the box. In case of success, lines 19 to 21 execute a natural split, otherwise, the box is stored in solutions (lines 23).

4.6 Detection of Box-Continuity

With an implementation of `HC_Filtering` like `HC4` [2], it is easy to detect box-continuity of a constraint during the filtering step of `Lazy_BoxSet`.

The projection operator of `HC4` must be slightly modified to update this property while exploring the syntax tree of a constraint. The rule is simple: before projecting a constraint c, the projection operator sets the box-continuity boolean of c to `true`. If a disjunction appears somewhere in the tree, this boolean is set to `false`.

Thus, in practice, detecting box-continuity is computationally insignificant and permits to dramatically reduce the number of calls to the general projection operator embedded in `Lazy_BoxSet`. This remark is relevant for all the problems, because at some point in the search, a majority of boxes are small enough for functions to be all monotonous. It is straightforward that with monotonous functions, detection of box-continuity always succeeds so that the projection operation is not costly.

4.7 Difference with Hyvönen's method

In [5], natural splitting is also used in a similar solving strategy. But an important difference makes our version much more powerful. In [5], no hull filtering is used before splitting and only box-continuous constraints are projected before instantiating variables. In a majority of non-linear problems, this extremely decreases the performances by generating an exponential number of "overlapping situations" [5], as this example illustrates:

Example 7. Let $P = (C, \{x_1, ..., x_{50}, y\}, D_{x_1} \times ... \times D_{x_{50}} \times D_y)$ be a NCSP.
$D_{x_1} = ... = D_{x_{50}} = [-2, 2]$ and $D_y = [1, 4]$.
C includes the following constraints:
$x_1^2 = y$
...
$x_{50}^2 = y$
Finally, C contains a trivially unsatisfiable constraint: $x_1^2 = -x_1^2$

We assume that constraints are treated in their declaration order. For all i $(1 \leq i \leq 50)$, projection of $x_i^2 = y$ over x_i gives $\{[-2, -1], [1, 2]\}$ so that `HC_Filtering` will not perform any reduction (bounds of $[-2, 2]$ are preserved). After these 50 unfruitful projections, `HC_Filtering` will fall on the last constraint, that makes the whole box inconsistent. So `Lazy_BoxSet` terminates almost immediately.

In contrast, as no constraint is box-continuous, the method in [5] will introduce a choice point for every variable x_i and deploy a search tree of 2^{50} leaves

before detecting inconsistency. A combinatorial explosion occurs. We observed this difference with simple problems of distance equations. In the general case, the proposed algorithm is more costly than our naive version.

Another drawback is that the domain of a variable must be divided statically into sub-intervals (the *actual application space*) where constraints are all box-continuous. This computation is only possible with primitive constraints.

5 Conclusion

We have tried to put an end to the question of arc-consistency with continuous domains, by showing precisely on a simple example that it is not applicable.

However, we have given a way to obtain the *box-set consistency*, i.e. all the arc-consistent sub-boxes of a problem, using a new splitting strategy called *natural splitting*.

The `Lazy_BoxSet` algorithm enforcing box-set consistency has been implemented, and so far, validated on toy problems. This implementation includes a projection operator for handling any type of constraints (not only primitive constraints). We will discuss about this crucial operator in a future paper, along with the conditions under which it can be applied.

Beyond this implementation, we believe that box-set consistency is a strong hence interesting property. We are currently investigating possible combinations of box-set filtering and interval analysis.

References

1. F. Benhamou. Interval constraint logic programming. In A. Podelski, editor, *Constraint Programming: Basics and Trends, LNCS no 910*, pages 1–21. Springer Verlag, 1995.
2. F. Benhamou, F. Goualard, L. Granvilliers, and J-F. Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244, 1999.
3. F. Benhamou, D. McAllester, and P. Van Hentenryck. Clp(intervals) revisited. In *International Symposium on Logic programming*, pages 124–138. MIT Press, 1994.
4. B. Faltings. Arc-consistency for continuous variables. *Artificial Intelligence*, 65, 1994.
5. E. Hyvönen. Constraint reasoning based on interval arithmetic—The tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.
6. O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles.* Phd thesis, University of Nice-Sophia Antipolis, 1994.
7. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
8. R. Moore. *Interval analysis.* Prentice-Hall, 1977.
9. D.L. Waltz. Understanding line drawings of scenes with shadows. *The Psychology of Computer Vision*, pages 19–91, 1975.