New Light on Arc Consistency over Continuous Domains

Gilles Chabert, Gilles Trombettoni, Bertrand Neveu

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

New Light on Arc Consistency over Continuous Domains

Gilles Chabert — Gilles Trombettoni — Bertrand Neveu

N° 5365

Novembre 2004

Thème SYM

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

New Light on Arc Consistency over Continuous Domains

Gilles Chabert* , Gilles Trombettoni† , Bertrand Neveu‡

Thème SYM — Systèmes symboliques
Projet COPRIN

Rapport de recherche n° 5365 — Novembre 2004 — 28 pages

Abstract: Hyvönen [9] and Faltings [7] 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 the first part of this report, 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 the so-called *box-set consistency*, and devise a generic method to enforce it. This method combines hull consistency filtering, interval union projection, and intelligent domain splitting.

In the second part, a concrete algorithm, derived from a lazy version of the generic method is proposed. It can be applied to any numerical CSP, and achieves box-set consistency providing that constraints are redundancy-free in terms of variables. This holds even if the problem is not interval-convex. The main contribution of our approach lies in the way we bypass the non-convexity issue, which so far was a synonym for either a loss of accuracy or an unbounded growth of label size. We prove the correctness of our algorithm and through experimental results, we show that, as compared to a strategy based on a standard bisection, it may lead to gains while never producing an overhead.

Key-words: constraint programming, interval arithmetics, arc consistency.

This report subsumes two papers, *New light on Arc Consistency over Continuous Domains* in the First International Workshop on Constraint Propagation and Implementation (2004) and *Box-Set Consistency for interval-based constraint problems*, in the Symposium on Applied Computing (2005).

* gilles.chabert@sophia.inria.fr
† trombe@sophia.inria.fr
‡ bertrand.neveu@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Nouvel éclairage sur l'arc-cohérence en domaine continu

Résumé : Les travaux de Hyvönen [9] et Faltings [7] ont montré que le recours aux unions d'intervalles dans les algorihmes de filtrage par propagation conduisait à des pertes d'efficacité inacceptables.

Une telle représentation des domaines était introduite dans le but d'appliquer la propriété d'arc-cohérence sur des problèmes continus. Cette propriété, bien connue en domaine fini, garantit que chaque valeur du domaine d'une variable possède un support dans le domaine des autres variables.

Dans la première partie de ce rapport, nous montrons qu'il ne peut pas exister de filtrage par arc-cohérence en domaine continu, si on se limite à la propriété telle quelle. Mais, paradoxalement, il est possible d'obtenir pour de nombreux problèmes une consistance plus forte, moyennant l'introduction de quelques points de choix. Nous définissons la propriété dite de *box-set cohérence* et donnons une méthode générique permettant de l'obtenir. Cette méthode s'appuye sur un filtrage par 2B-consistance, la projection d'unions d'intervalles et une technique particulière de bissection.

A partir d'une version paresseuse de cette méthode, nous élaborons dans la seconde partie un algorithme concret dont nous détaillons certains rouages techniques. Ses propriétés théoriques sont ensuite étudiées, notamment les conditions sous lesquelles l'arc-cohérence est effectivement garantie. Nous montrons en particulier qu'elle l'est pour des problèmes non-convexes, au sens "intervalles", pour autant que les contraintes soient sans occurrence multiple de la même variable.

Le principal intérêt de notre approche réside dans la façon dont nous contournons le problème des fonctions non convexes qui, jusqu'ici, impliquait soit de faire des approximations, soit de gérer des unions d'intervalles de taille non bornée.

Nous avons effectué une première évaluation expérimentale de notre approche. Nous observons d'une part que le fait d'intégrer un filtrage par box-set cohérence dans un algorithme de recherche de solution ne détériore jamais les performances, et d'autre part, qu'il peut y avoir dans certains cas des gains allant jusqu'à 20%.

Mots-clés : programmation par contraintes, arithmétique d'intervalles, arc-cohérence.

1 Introduction

Constraint programming (CP) is a complementary approach of interval-based numerical methods to solve systems of equations over the reals. Instead of approximating solutions with mathematical reasoning on the entire system, equations are treated as independent compatibility relations (called *constraints*) between variables. In this perspective, domains are filtered by removing values for which a relation cannot be satisfied individually. On the one hand, these methods make hardly use of semantics, but on the other hand, they are general-purpose. A mix of CP and interval analysis has given rise to the best solvers for satisfying or optimizing constraint systems.

When the domains of the variables are stable for all the constraints, i.e., when all the values can be used to satisfy any constraint, we see that we cannot go further with "local" reasoning, and we talk about *arc consistency*. A lot of algorithms have been designed to enforce *arc consistency* in discrete constraint systems (where domains have a finite number of values). In contrast, arc consistency has never been applied successfully with real variables. 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 the so-called *box-set consistency*, a stronger property which yields a set of arc consistent boxes (i.e., products of intervals).

A generic method with two variants is proposed for box-set consistency, followed by a concrete implementation of the lazy one.

The report 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 the box-set consistency and show related properties. A generic method that enforces this consistency is given in Section 4.

In Section 5, we show in practice how to enforce box-set consistency for a class of projection-friendly problems. Then, we expound in Section 6 the theoretical properties of this algorithm, and eventually deal with performances, in Section 7.

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 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 report, 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 2.1 (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 sub-nodes in the tree.

A major approach for local filtering is the well-known *hull consistency*, obtained by a Waltz-like propagation algorithm [16]. 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 [14].
- **Propagation**: Propagate reductions over the other variables linked to x by another constraint c'.

2.2 Projections

Local filtering techniques are based on an interval narrowing operator, called *projection*, that computes compatible values for different variables linked by a constraint.

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 2.2 (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 2.1 $c : x + y = z$
$\Pi_x^c : D_x \times D_y \times D_z \longrightarrow (D_z \ominus D_y) \cap D_x$

[1] Also called *Solution function* in [9].

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

In this case, the *hull consistency* algorithm [1] presented below (also known as *2B consistency* [12]) computes the enclosing interval of the union immediately (i.e., $[-2, 2]$) therefore losing track of the "gap" between intervals.

The set of values returned by a projection may be either an interval (example 2.1) or a union of intervals (example 2.2). 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.

In this report, 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 2.3 (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 2.1 *In this report, 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*:

[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 report with definitions.

Definition 2.4 (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
```

The idea of this algorithm originates from Waltz [16] and was applied first over finite domain constraints under the acronym `AC3` [13] to obtain arc consistency. With intervals, `HC3` [3] introduces a decomposition of the system into *primitive constraints* (this term is defined further) for which projections are known, and `HC4` [2] is an upgraded version of `HC3` that produces the same result sparing decomposition. Both enforce hull-consistency.

3 A property stronger than arc consistency

In this section, we introduce a new kind of consistency, from a theoretical point of view. The rest of the report will be devoted to the way it can be enforced. Contrary to arc consistency, it can be obtained with a reasonable space complexity 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 3.1 (AC Part) *The AC part of a NCSP is the maximal arc consistent sub-NCSP, for the order relation of inclusion (see definition 2.1).*

Example 3.1 *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, like 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 3.2 *Let* $P = (\{c_1, c_2\}, \{x, y\}, B)$ *be the following NCSP:*
$B = D_x \times D_y = [1, 9] \times [1, 9]$
$(c_1): \quad (\frac{3}{4}(x - 5))^2 = y$
$(c_2): \quad y = x$

Lemma 3.1 *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[3] 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: $D_x \leftarrow \Phi(D_y)$ and $D_y \leftarrow D_x$. 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 $X_2 \times Y_2$.

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

Property 3.1 $\forall n \geq 1, \quad X_n \subset X_{n-1}$ *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 $X_{n+1} \subset X_n$. ▲

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

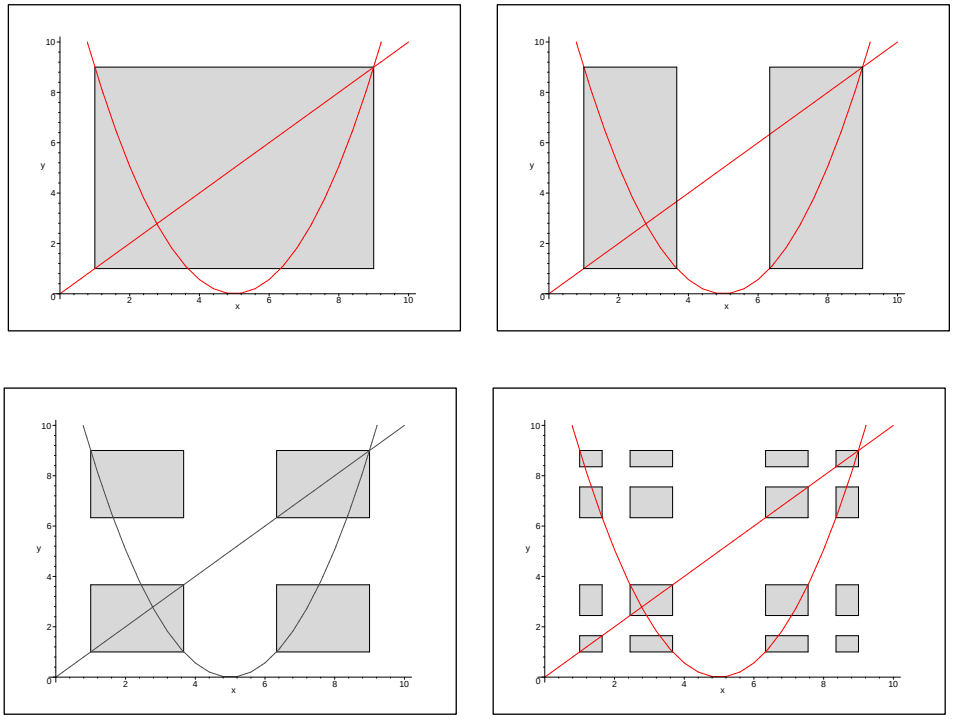


Figure 1: First steps of `AC` filtering

Property 3.2 *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 fall 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[4]. 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 3.1, we can check that intervals of X_{n+1} and Y_{n+1} are included in $[1,9]$. ▲

Property 3.3 *Bounds of intervals are always maintained in domains. That is to say, if $[a,b]$ is an interval of X_n[5], then $\forall m \geq n$, a and b are interval bounds of X_m.*

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 $m \geq n$, then bounds of X_{n+1} will also be maintained for all $m \geq n+1$ since they take support on bounds of Y_n, i.e., X_n, and they are included in X_n (property 3.1). ▲

Now, property 3.1 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 3.2 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 3.3), 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 arbitrary number of iterations. In a nutshell :

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

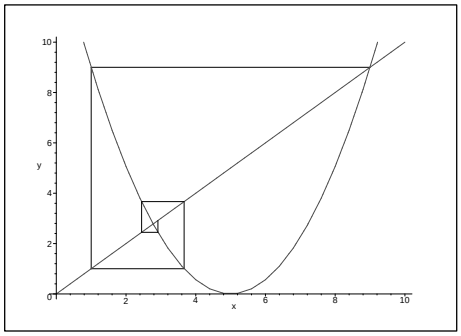
Sufficient condition

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

[4] F_1 and F_2 are optimal

[5] 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 $[a,b] \subset X_n$

We prove easily that $u_n \rightarrow \frac{25}{9}$ when $n \rightarrow +\infty$. This convergent sequence 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 3.2: 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 3.1.
Remark: In the AC part of P, intervals can be punctual.

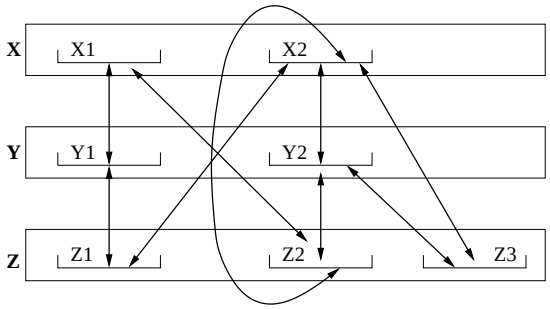
3.2 Box-set consistency

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 2.3 and remark 2.1).

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 pair-wisely 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.



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

Definition 3.2 (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.

The following (real) example reinforces the fact that box-set consistency is stronger than arc consistency:

Example 3.3 $D_x = D_y = D_z = [-2, 2]$ $D_w = [1, 4]$
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 consistency. It suffices to consider this NCSP:

Example 3.4 $D_x = [0, 2]$ $D_y = [0, 2]$ $D_z = [0, 2]$
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 $\{(1, 1, 1)\}$.

We will check in Section 6.3 that the size of the box-set consistency (i.e., the number of boxes) is bounded, contrary to arc consistency.

4 Natural splitting

In this section, we show a method for enforcing box-set consistency. This method 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 natural split 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 *interval-convexity*. We begin by introducing this notion and give the method afterwards.

4.2 Interval-Convexity

Interval-convexity is the key property of our approach. It has already been defined in [4]. A constraint c is *interval-convex* on a given box when projections of c do not create gaps, i.e., when the result set of a projection of c over any variable contains a single interval. Formally:

Definition 4.1 (Interval-Convexity) *Let c be a constraint relating variables $x_1, ..., x_k$. Let B be a box.*
c is interval-convex on $B \Longleftrightarrow \forall i \ (1 \leq i \leq k) \ |\Pi^c_{x_i}(B)| = 1$.

Interval-convexity is not related to the continuity of 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 interval-convex 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 interval-convex 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 interval-convexity 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 interval-convex on the current box.

The following procedure is the first version of our generic 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 Completeness

Proposition 4.1 *Let P be a NCSP.*
`Naive_BoxSet` *gives the box-set consistency of P.*

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 interval-convex on B. But B is also hull-consistent so the following lemma applies to B:

Lemma 4.1 *If every constraint is interval-convex on a box B then:*
B is hull-consistent $\iff$ 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: $D_x = \Box\Pi_x^c(B)$. As c is interval-convex, $\Box\Pi_x^c(B) = \Pi_x^c(B)$ 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, the leaves in the search tree of `Naive_BoxSet` are arc consistent. Now, we need to prove that they are also maximal:
Let B be the initial box. Let B' be the box obtained by `Naive_BoxSet` just before the first natural split, and B_1,...,B_n the child boxes obtained just after. It is clear that the box-set consistency of B is also the box-set consistency of B'. We need to prove that the box-set consistency of B' is the union of the box-set consistencies of B_1,...,B_n. Assume that it is not. We can find a box X in the box-set consistency of a child box, for instance B_1, that is not maximal. Therefore X can be enlarged, i.e., an "adherent" value can be added in the domain of a variable (a value stuck to X). As B' and B_1 differ only in the domain of the split variable (say x), by a simple reasoning on propagation, the added value requires an extra value in the domain of x. But D_x cannot be enlarged, because an adherent value of D_x would be inside the gap, and necessarily inconsistent. Hence, by a straightforward induction, the box-set consistency of the initial box is the union of the box-set consistencies of the leaves. We have seen that leaves are arc consistent, so they are maximal.

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 interval-convex or not (we will see in Section 5.4 an easy way to do that). 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 interval-convex. 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 natural 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
    gap ← false
8:  while (not gap) and (C' ≠ ∅) do
      pop c from C'
10:   if c is not interval-convex then
        V' ← the set of variables in V related by c
12:     while (not gap) and (V' ≠ ∅) do
          pop x from V'
14:       U ⟵ Π_x^c(B)
          if |U| > 1 then
16:         gap ← true

18: if gap 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 Difference with Hyvönen's method

In [9], natural splitting is also used in a similar solving strategy. But an important difference makes our version much more powerful. In [9], no hull filtering is used before splitting and only interval-convex 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" [9], as this example illustrates:

Example 4.1 *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:
$x_1^2 = y \quad ... \quad x_{50}^2 = y$
Finally, C contains a trivially unsatisfiable constraint: $x_1^2 = -1$

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 interval-convex, the method in [9] 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 subintervals (the *actual application space*) where constraints are all interval-convex. This computation is only possible with primitive constraints (see next section).

5 Practical issues

`Lazy_BoxSet` is a theoretical method because it is based on a projection operator, and nothing is said about the way to implement it. This operator is called at two different steps: during the hull consistency filtering, and during the gap search.

We are going to present in this section a way to work out this operator, what *de facto* will give us a ready-to-use version of a box-set filtering algorithm. We will say that a projection operator is *exact* when it respects the definition 2.2.

5.1 Preliminary notions

The implementation proposed achieves box-set consistency only if constraints are redundancy-free in terms of variables (but this holds even if the problem is not interval-convex). Therefore, we need to consider various levels of complexity for the problems to state the correctness of our algorithm.

- **NCSPs with primitive constraints only**. A *primitive* constraint is a basic binary or ternary relation such as $z = x + y$. As the projections are mathematically known, they can be hard coded. The set of primitive constraints is given in Figure 2 below.
- **Simple NCSPs**. A constraint c is *simple* if the syntax of c does not include multiple occurrences of the same variable. For instance, $z = (x+y)^2$ is simple but $z = x^2 + y^2 + 2xy$ is not. A simple constraint is a combination of several primitive constraints. A NCSP is *simple* if all the constraints are simple.
- **General case**. Any constraint that is not simple falls into the general case. Intuitively, it is a hard problem to infer projections from the syntax of a constraint when variables occur more than once.

$p_1(x,y):$	$x = y$	$p_6(x,y,z):$	$z = x * y$
$p_2(x,y,z):$	$z = x + y$	$p_7(x,y,z):$	$z = x/y$
$p_3(x,y,z):$	$z = x - y$	$p_8(x,y,a):$	$y = x^a$
$p_4(x,y):$	$y = exp(x)$	$p_9(x,y):$	$y = cos(x)$
$p_5(x,y):$	$y = ln(x)$	$p_{10}(x,y):$	$y = sin(x)$

Figure 2: Primitive constraints

5.2 Overview

For performance reasons, we chose `HC4` [2] for our implementation of hull consistency. But the projection operator embedded in `HC4`, called `HC4Revise`, is exact only for primitive constraints. Despite this restriction, we will see that it is possible to obtain the box-set consistency of simple NCSPs, providing that projections for the gap search are exact. That is why we use a different operator for the gap search, called `TAC`.

Procedure 4 `HC4+TAC`(NCSP (C, V, B), in-out `results`)

```
2:  HC4((C, V, B))

4:  if B is empty then
      return
6:  C' ← C
    gap ← false
8:  while (not gap) and (C' ≠ ∅) do
      pop c from C'
10:   if not (interval-convex(c)) then
        V' ← Variables(c, V)
12:     while (not gap) and (V' ≠ ∅) do
          pop x from V'
14:       U ⟵ TAC(c, x, B)
          if U ⊊ D_x then
16:         gap ← true

18: if gap then
      for all intervals I in U do
20:     D_x ⟵ I          // D_x is the domain of x in B
        HC4+TAC((C, V, B), results)
22: else
      add B to results
```

Hence, we do not use the same operator in both places we have mentioned (hull consistency filtering and gap search), and this is why our algorithm is sumed up in the acronym `HC4+TAC`.

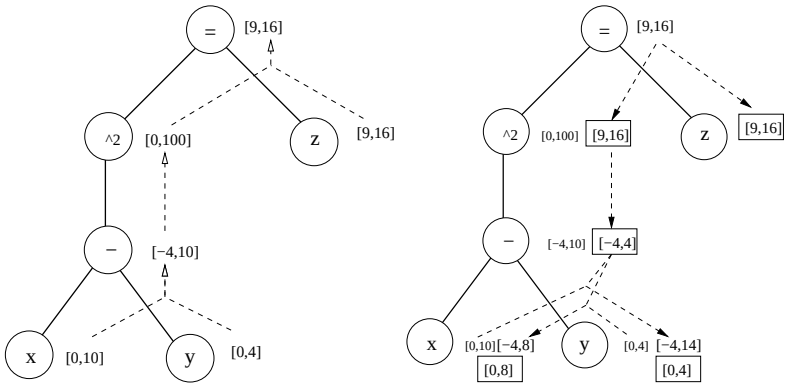
Note that line 15 hides a subtlety: `gap` may be true either when U is a union (as in the original algorithm), or when bounds of D_x can be narrowed. The latter case arises due to the limitation of `HC4Revise` evoked before. We will get back to this issue in 6.2. The rest of this section describes the projection operators, `HC4Revise` and `TAC`. Both are described into [2].

5.3 HC4Revise

First, let us summarize how `HC4Revise` works through an example. See [10] and [2] for a complete description.

Let $c : (x - y)^2 = z$ be a constraint relating $x \in [0, 10], y \in [0, 4]$ and $z \in [9, 16]$. Constraints are represented by their abstract syntax tree, where each node stands for a sub-expression.

Figure 3(a) depicts the first step of the retro-propagation algorithm, an upward traversal of the tree that computes an evaluation of every sub-expression in the tree, using interval arithmetics.



(a) upward propagation (b) downward propagation

Figure 3: `HC4Revise`

Figure 3(b) shows the second step, the downward traversal of the tree that computes the projection over every sub-expression. Actually, a node is linked to its child nodes by a primitive constraint, and this step simply consists in applying cascading projections of primitive constraints.

Finally, we notice that `HC4Revise` has reduced the domain of x from $[0, 10]$ to $[0, 8]$.

Except with primitive constraints, `HC4Revise` does not compute an exact projection, as defined in Section 2, but a conservative approximation. Indeed, when a disjunction (i.e., a union of intervals) occurs during the second step, the hull is computed so that inconsistent values are introduced in place of gaps. In our example, the exact projection of $[9, 16]$ over the node labeled with "–" is $\{[-4, -3], [3,\ 4]\}$, and only the hull $[-4, 4]$ is kept.

5.4 Detection of Interval-Convexity

By using `HC4` for the implementation of `HC_Filtering`, it is easy to detect interval-convexity of a constraint.

`HC4Revise` must be slightly modified to update this property while exploring the syntax tree of a constraint. The rule is simple: before handling a constraint c, `HC4Revise` sets the interval-convexity boolean of c to `true`. If a disjunction appears somewhere in the tree, this boolean is set to `false`.

Thus, in practice, detecting interval-convexity is computationally insignificant and permits to dramatically reduce the number of calls to the (costly) operator `TAC`. 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 interval-convexity always succeeds so that the projection operation is not costly.

5.5 TAC

The algorithm `TAC` (**T**ree **A**rc **C**onsistency) computes an exact[6] projection with simple NCSPs. It is the union-variant of `HC4Revise`. The exactly same backward propagation idea is used, except that interval unions are stored in place of intervals. This variant has already been evoked in [2].

We preclude from our discussion the particular case of trigonometric functions with an infinite number of acceptable periods, such as $sin(1/x)$ with $0 \in D_x$. We will see later a way to tackle such constraints.

`TAC` is exactly `HC4Revise` except that unions of intervals are authorized. In our example, the node labeled with "-" is assigned a union of intervals : $\{[-4, -3], [3,\ 4]\}$. Projecting this union over x leads to a union for x, as shown in Figure 4. Note that a gap cannot appear with primitive constraints in the left column of Figure 2 (functions are monotonous), as opposed to the constraints in the right column. Note also that, except with p_7, only the downward propagation requires union labeling.

By limiting the scope of propagation to a single constraint, an infinite loop as described in 3.1 cannot occur. Indeed, a necessary condition for such a loop is the presence of cycles in the constraint network. Even more, the label size (i.e., the number of intervals used to represent domains) is bounded, this bound being inherent to the problem and easily computable (see [9]).

[6]This term is employed regardless of the rounding due to the machine representation of reals (not considered here).

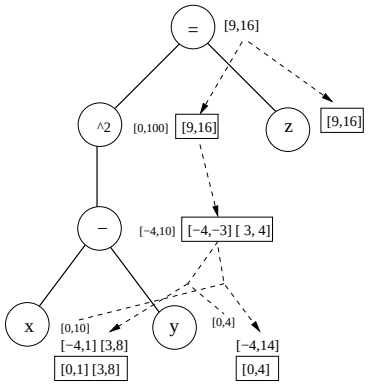


Figure 4: Downward step of TAC

Remark: We also use TAC in presence of multiple variable occurrences in a same constraint. Each occurrence is treated as a different and independent variable. Of course, exactness of the output is no more guaranteed. In Section 6, we explain what we can expect from TAC in this case.

5.6 Dealing with infinity of disjunctions

In this case, we need to introduce a bound for the label size, say MAX. If at some point the label size of a projection result exceeds MAX, we simply merge some of the intervals. The projection is not exact any longer, but as long as two intervals are isolated, the other ones can be merged and their split is just postponed. If the system has a finite number of solutions, the process will end. This implementation trick prevents the solver from failing in case of overflow. In all our benchmarks, we never observed an overflow with a bound set to 10.

6 Theoretical properties

In this section, we clarify the properties of our algorithm because projections are not exact in all the situations. We start from primitive constraints, go on with simple NCSPs and finally deal with the general case.

Proposition 6.1 *Let P be a NCSP.*
If constraints in P are all primitive, then HC4+TAC *enforces the box-set consistency of P.*

Proof: It is shown in 4.4 that Naive_BoxSet, hence Lazy_BoxSet, solves the box-set consistency of a NCSP. Then, since both TAC and HC4Revise compute an exact projection of a primitive constraint, the box-set consistency is ensured by HC4+TAC. ▲

6.1 Simple NCSPs

The previous result can be extended to systems with arbitrary complex expressions, assuming that variables appear at most once inside a same constraint.

Proposition 6.2 *Let P be a simple NCSP.*
`HC4+TAC` *enforces the box-set consistency of P.*

The proof is not as straightforward as for the proposition 6.1 because `HC4Revise` does not always compute exact projections (see 6.1.3), and consequently hull consistency is not guaranteed by `HC4` anymore. But fortunately, the situations where `HC4` fails to achieve hull consistency are intermediate steps of the general algorithm, so that this limitation does not prevent `HC4+TAC` from giving the box-set consistency.

First, we are going to show that, in this case, `TAC` still computes exact projections.

6.1.1 Exactness of `TAC`

Let c be an arbitrary constraint. We call *decomposition* of c the sub-system of primitive constraints equivalent to c.

Example 6.1 (Decomposition) *$c : (x-y)^2 = z$ is equivalent to a sub-system S of 3 primitive constraints relating 5 variables (x, y, z, w, t):*

$$c : (x-y)^2 = z \Longleftrightarrow (S) \begin{cases} p_3(x, y, w) \\ p_8(w, t, 2) \\ p_1(t, z) \end{cases} \quad (1)$$

Proposition 6.3 *Let c be a simple constraint.*
`TAC` *computes exact projections of c.*

Proof:
Let S be the decomposition of c.

1- The constraint network of S is a tree (the same as the syntax tree of c).

2- Each time a union is computed for a node in `TAC`, there is an equivalent projection in S over the implicit variable (like w and t in example 6.1) representing this node. Initial domains of implicit variables are $]-\infty, +\infty[$.

3- The projection of a constraint in S over a given variable is exact (proposition 6.1), i.e., every value in the resulting domain of this variable has a support in the domains of the other variables. Such projection is equivalent to a step of *directed arc consistency* in finite domain [5].

4- Faltings showed in [7] that a two-step *directed arc consistency filtering* applied on a tree-structured graph leads to an arc consistent labeling. Each step of TAC is equivalent to a step of directed arc consistency, from the leaves of the tree (the variables of c) up to the root and in the other way around.

5- Arc consistency of a tree-structured graph is equivalent to global consistency [8]. So TAC gives the global consistency of S. In other words, every value in the domain of a variable in c belongs to a solution of S, i.e., satisfies the constraint c, since c and S are semantically equivalent. Therefore, global consistency of S is equivalent to arc consistency of c. This justifies the name (**T**ree **A**rc-**C**onsistency) of this method. Hence, this two-step process computes exact projections over all the involved variables.

▲

6.1.2 Proof of proposition 6.2

Consider the leaves in the search tree of HC4+TAC.

Leaves are arc consistent: If no disjunction occurs during a call to HC4Revise, no intermediate hull is computed and HC4Revise behaves rigorously like TAC. Assume a disjunction occurs during a call to HC4Revise. The constraint is marked as "non interval-convex", and it will be projected by TAC. But TAC cannot perform any reduction with this constraint, since we are dealing with a leaf (any reduction entails a subsequent call to HC4+TAC). Therefore, any call to HC4Revise is necessarily equivalent to a projection by TAC. Since we have proven in 6.1.1 that TAC is exact, once the fixpoint is reached, arc consistency is achieved.

Leaves are maximal: The proof is the exact copy of the one given in 4.4.

6.1.3 Remark: The difficulty with HC4

HC4Revise does not always compute an exact projection, because of intermediate approximations while projecting over implicit variables. The consequence is that HC4 does not enforce the hull consistency of the system anymore, but only the hull consistency of the decomposition of the system [2].

Example 6.2 *Let P be the following CSP:*
$D_x = [0,1]$, $D_y = [-1,1]$
$(x \times y)^2 = 1$

Let P' be the decomposition of P, i.e. :
$D_x = [0,1]$, $D_y = [-1,1]$, $D_w =]-\infty,+\infty[$
$x \times y = w$
$w^2 = 1$

P is not hull consistent because the bound 0 for x has no support in y, and it is easy to check that HC4 cannot reduce this bound. Indeed, the projection over w (the subexpression $x \times y$) results in the interval $[-1,1]$ that includes the bad support $w = 0$ for $x = 0$. On the

other hand, hull consistency of P' is achieved with an *extension* of B, i.e., a box $B' = B \times D'_w$. This extension is $[0,1] \times [-1,1] \times [-1,1]$.

So we could have expected HC4+TAC to enforce the box-set consistency of the decomposition of a problem. This cannot be true because arc consistency of a system is a weaker property than arc consistency of its decomposition.

Example 6.3 *Let P be the following CSP:*
$D_x = [0,4]$, $D_y = [0,4]$
$(x-y)^2 = 4$

Let P' be the decomposition of P, i.e. :
$D_x = [0,4]$, $D_y = [0,4]$, $D_w =]-\infty, +\infty[$
$x - y = w$
$w^2 = 4$

P is arc consistent. But arc consistency of P' cannot be achieved with an extension of B, i.e., with $D_x = [0,4]$ and $D_y = [0,4]$. Indeed, the box-set consistency of P' includes two arc consistent boxes : $\{[0,2] \times [2,4] \times [-2,-2]$ and $[2,4] \times [0,2] \times [2,2]\}$. This is not a big surprise since, by decomposing a system, it becomes possible to split the domains of the implicit variables (w) and therefore, to obtain smaller boxes.

This hybrid situation can be summarized with this pseudo-formula:

(1) **hull consistency $\Longrightarrow$ hull consistency of the decomposition**
(2) **arc consistency $\Longleftarrow$ arc consistency of the decomposition**

See [6] for (1). The flip in the implications would mean *a priori* that a box returned by HC4+TAC has not been filtered enough to be hull consistent (regarding P), and not been split enough to be arc consistent (regarding P'). However, we have seen how to fix the problem with a recursive call condition (see line 15 of HC4+TAC) that ensures that the fixpoint of TAC is met for the leaves. Therefore the box-set consistency of P is obtained.

6.2 General Case

To describe the property of HC4+TAC in this case, follows the definition of another kind of transformation called *renaming* [6], where each occurrence of the same variable is substituted by a new variable[7].

Example 6.4 (Renaming)

$$c : (x-y)^2 - x = 0 \Longleftrightarrow (S) \begin{cases} (x-y)^2 - z = 0 \\ z = x \end{cases} \tag{2}$$

x and z in (S) are the aliases of x in c.

Note that renaming produces an intermediate system between the original one and the decomposition, in terms of size.

[7] It seems that no dedicated terms exist in literature to distinguish *renaming* from *decomposition*. Depending on the paper, *decomposition* may either refer to the first or the latter.

Proposition 6.4 *Let P be a NCSP. Let P' be the renaming of P* `HC4+TAC` *applied on P enforces the box-set consistency of P'.*

Thus, the box-set consistency cannot be enforced by `HC4+TAC` in the general case.

Proof: With multiple occurrences of variables, `HC4+TAC` behaves as if it was applied on the renaming of the system. For instance, the first operation made by `HC4Revise` and `TAC` with a constraint is to assign to the aliases of a same variable x the domain of x. So, if aliases of x are noted $z_1, ..., z_n$, the first step of these operators is equivalent to a projection of the non-root constraints $z_i = x$ over the aliases z_i in the renaming. One precaution however with multiple occurrences : we must push back the current constraint in the propagation queue after a call to `HC4Revise`.

So, `HC4+TAC` provides the same result as if it was applied on the renaming of the problem, which is a simple NCSP. But we know that `HC4+TAC` applied on a simple problem enforces box-set consistency (proposition 6.2), so `HC4+TAC` enforces the box-set consistency of the renaming. ▲

6.3 Number of boxes of a box-set consistent problem

For a given variable v, the number of intervals where projections are all monotonous 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 $(p \times a)^n$, since monotonicity is a stronger property than interval convexity. In Hyvönen's terminology [9], this number bounds the size of the *global application space*.

p is known for any primitive constraint. For instance, with $y = x^2$, we have $p = 2$ and this value corresponds to the number of implicit functions ($-\sqrt{y}$ and $+\sqrt{y}$).

We have seen before that the box-set consistency of the decomposition does not imply the box-set consistency of the original problem. But the following rule applies anyway: *The projection of a simple constraint cannot produce more intervals than its decomposition.* This is just a consequence of `TAC`'s exactness.

Therefore, a bound for p can also be computed for a simple constraint by observing primitives functions embedded in the syntax of the constraint, and by combining their associated p-values:

Example 6.5 $sin(5\pi/8 \times (x-y)^2) = 0.75$
With $D_x = D_y = [-1, 1]$, the value of p obtained (for both x and y) is $2 \times 3 = 6$, because :

- *Estimation of $(x - y)$ is $[-2, 2]$ and the square function has 2 monotonous branches on this domain.*
- *Estimation of $5\pi/8 \times (x-y)^2$ gives $[0, 5\pi/2]$ and the sine function has 3 monotonous branches on this domain.*

The bound can be adjusted with symbolic handling. For instance, if (c) is $(((x^2)^2)^2)^2 = y$, then $p \leq 1 \times 2 \times 2 \times 2 \times 2 = 8$. But by rewriting this constraint into the equivalent form $x^8 = y$ it follows that, in fact, $p \leq 2$.

On the contrary, no result seems easy to establish in the general case since the rule above is clearly false when variables have multiple occurrences (consider for instance $y = x^3 - x$ where $p = 3$. The renaming is made of two monotonous constraints, $z = x^3$ and $y = z - x$). We have not found straightforward bounds.

Remark 6.1 *The bound $O(p \times a)^n$ is exponential, but it can be strongly reduced by the dependencies between variables. For instance, in example 6.5, by adding the constraint $x + y = 0$, the box-set consistency of the problem includes 6 boxes, whereas the computed bound is $6^2 = 36$.*

7 Practical Time Complexity

In practice, box-set consistency can be used to find solutions of a NCSP. Instead of collecting the resulting boxes (see line 23 in `HC4+TAC`), we use them in a depth-first process as new choice points. In this way, the advantage of box-set consistency is twofold: First, the property itself may be exploited, and this approach is still under research. Second, as natural splitting is driven by the semantics of the problem, it is sharper and often more efficient than other bisection heuristics (e.g., round-robin or largest-domain first).

On 20 problems found in [15] and `ALIAS` solver homepage[8], We have compared `HC4` and bisection, with `HC4+TAC` and bisection. In 15 of them, computation times are equivalent; in 3 others `HC4+TAC` provides a gain up to 10%. A 20% gain was obtained on the Broyden banded function problem [15], and on the Sierpinski's distance equations problem, which include both 30 nonlinear equations. This ratio is maintained even by introducing higher order consistencies [11] in the solving process.

8 Conclusion

We have tried to clear up 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*.

We have shown in practice how to enforce box-set consistency (hence arc consistency), without spoiling performances. The keystone of this implementation is an operator that manages gaps, and the number of calls to this costly operator is minimized by using a standard hull consistency algorithm as a pre-filtering process.

The main contribution was to design in detail this lazy algorithm and establish precisely the properties. We have proven that box-set consistency is obtained with simple NCSPs,

[8] `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches`

including non interval-convex constraints. We obtain a relaxation of the box-set consistency in the general case.

Beyond this implementation, we believe that the 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] F. Benhamou and W.J. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, 32:1–24, 1997.

[5] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[6] F. Delobel, H. Collavizza, and M. Rueher. Comparing partial consistencies. In *Reliable Computing*, pages 213–228. Kluwer, 1999.

[7] B. Faltings. Arc-consistency for continuous variables. *Artificial Intelligence*, 65, 1994.

[8] E. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[9] E. Hyvönen. Constraint reasoning based on interval arithmetic—The tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.

[10] Y. Lebbah. *Contribution à la résolution de contraintes par consistance forte.* Phd thesis, University of Nantes, 1999.

[11] O. Lhomme. Consistency techniques for numeric csps. In *Proc. of the 13th IJCAI*, pages 232–238, 1993.

[12] O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles.* Phd thesis, University of Nice-Sophia Antipolis, 1994.

[13] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[14] R. Moore. *Interval analysis.* Prentice-Hall, 1977.

[15] J. Moré, B. Garbow, and K. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Softw.*, 7(1):17–41, 1981.

[16] D.L. Waltz. Understanding line drawings of scenes with shadows. *The Psychology of Computer Vision*, pages 19–91, 1975.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399