# Filtering Numerical CSPs Using Well-Constrained Subsystems

Ignacio Araya, Gilles Trombettoni, Bertrand Neveu

INRIA, Université de Nice-Sophia, CERTIS
{Ignacio.Araya,Gilles.Trombettoni,Bertrand.Neveu}@sophia.inria.fr

**Abstract.** When interval methods handle systems of equations over the reals, two main types of filtering/contraction algorithms are used to reduce the search space. When the system is well-constrained, interval Newton algorithms behave like a global constraint over the whole $n \times n$ system. Also, filtering algorithms issued from constraint programming perform an AC3-like propagation loop, where the constraints are iteratively handled one by one by a *revise* procedure. Applying a revise procedure amounts in contracting a $1 \times 1$ subsystem.
This paper investigates the possibility of defining contracting well-constrained subsystems of size $k$ ($1 \leq k \leq n$). We theoretically define the *Box-k-consistency* as a generalization of the state-of-the-art Box-consistency. Well-constrained subsystems act as *global constraints* that can bring additional filtering w.r.t. interval Newton and $1 \times 1$ standard subsystems. Also, the filtering performed inside a subsystem allows the solving process to learn interesting multi-dimensional branching points, i.e., to bisect several variable domains simultaneously. Experiments highlight gains in CPU time w.r.t. state-of-the-art algorithms on decomposed and structured systems.

## 1 Introduction

When interval methods handle systems of equations over the reals, two main types of filtering/contraction algorithms are used to reduce the search space. When a system contains $n$ unknowns/variables constrained by $n$ equations, interval Newton algorithms behave like a global constraint over a linearization of the whole $n \times n$ system. Filtering algorithms issued from constraint programming handle $1 \times 1$ subsystems (one variable involved in one constraint) in an AC3-like propagation loop.

This paper investigates the possibility of filtering $k \times k$ subsystems, where the size $1 \leq k \leq n$. After introducing in Section 2 the necessary background about intervals, we define in Section 3 the Box-k-consistency achieved by our new algorithm. This partial consistency generalizes the well-known Box-consistency [2]. Due to the large amount of subsystems in a constraint system, we explain in Section 5 the criteria used to compute the Box-k-consistency in only certain subsystems that are made of equalities, connected and well-constrained. These subsystems are managed like *global constraints* [16, 10] for enhancing the

filtering power. We detail in Section 4 the filtering (revise) procedure that filters one subsystem and makes it box-k consistent. The procedure expands a local search tree whose choice points are limited inside the subsystem, and uses a local interval Newton. This revise procedure has common points with the algorithm proposed in [5]. Their algorithm also performs a tree search where every node is filtered, before returning an outer approximation of the obtained sub-boxes. But it is applied to the whole system of equations and not to subsystems.

Section 5 details how the local search trees built inside subsystems allow a solving strategy to learn interesting multi-dimensional choice points in the global search tree, i.e., to bisect several variable domains simultaneously. These multi-dimensional branching points are called *multisplits*. Promising experiments highlight the benefits of our approach for *decomposed* and *structured* NCSPs.

## 2 Background

The algorithms presented in this paper aim at solving systems of equations or, more generally, numerical CSPs.

**Definition 1** *A* **numerical CSP** *(NCSP)* $P = (X, C, B)$ *contains a set of constraints* $C$ *and a set* $X$ *of* $n$ *variables. Every variable* $x_i \in X$ *can take a real value in the interval* $[x_i]$ *and* $B$ *is the cartesian product (called a* **box***)* $[x_1] \times ... \times [x_n]$*. A solution of* $P$ *is an assignment of the variables in* $X$ *satisfying all the constraints in* $C$*.*

Since real numbers cannot be represented in computer architectures, note that the bounds of an interval $[x_i]$ should actually be defined as floating-point numbers. Most of the set operations can be achieved on boxes, such as inclusion and intersection. An operator `Hull` is often used to compute an outer approximation of the union of several boxes.

**Definition 2** *Let* $S = \{[b_1], ..., [b_n]\}$ *be a set of boxes corresponding to a same* $n$*-set of variables.*
*We call* **hull** *of* $S$*, denoted by* `Hull`$(S)$*, the minimal box including* $[b_1], [b_2], ..., [b_n]$*.*

To find all the solutions of an NCSP with interval-based techniques, the solving process starts from an initial box representing the search space and builds a search tree. The tree search **bisects** the current box, that is, **splits** on one dimension (variable) the box into two sub-boxes, thus generating one choice point. At every node of the search tree, filtering (also called *contraction*) algorithms reduce the bounds of the current box. These algorithms comprise **interval Newton** algorithms issued from the numerical analysis community [8, 13] along with contraction algorithms issued from the constraint programming community. The process terminates with **atomic boxes** of size at most $\epsilon$ on every dimension.

The new contraction algorithm presented in this paper generalizes the famous `Box` algorithm that can enforce the *Box-consistency* property [2] defined as follows:

**Definition 3** *An NCSP $(X, C, B)$ is* **box-consistent** *if every pair $(c, x)$ is box-consistent ($c \in C$, $x \in X$ and $x$ is one of the variables involved in $c$).*

*Consider a pair $(c, x)$, where $c(x, y_1, ..., y_a) = 0$ is an equation of arity $a+1$.*[1] *Let $c'$ be the equation $c$ where the variables $y_i$ are replaced by the current interval in $B$: $c'(x) = c(x, [y_1], ..., [y_a]) = 0$. The pair $(c, x)$ is box-consistent if:*

- $0 \in c'([\underline{[x]}, +]) = c([\underline{[x]}, +], [y_1], ..., [y_a])$;
- $0 \in c'([-, \overline{[x]}]) = c([-, \overline{[x]}], [y_1], ..., [y_a])$.

*$\underline{[x]}$, resp. $\overline{[x]}$, denotes the lower bound, resp. the upper bound, of $[x]$. $[\underline{[x]}, +]$ denotes the tiny interval (of one u.l.p. large*[2]*) bounded by $\underline{[x]}$ and the following float. $[-, \overline{[x]}]$ denotes the tiny interval bounded by $\overline{[x]}$ and the float preceding $\overline{[x]}$.*

In practice, the `Box` algorithm performs an AC3-like propagation loop. For every pair $(c, x)$, it reduces the bounds of $[x]$ such that the new left (resp. right) bound is the leftmost (resp. rightmost) solution of the univariate equation $c'(x) = 0$. Existing *revise* procedures use a *shaving* principle to narrow $[x]$: Slices $[s_i]$ inside $[x]$ with no solution are discarded by checking whether $c([s_i], [y_1], ..., [y_a])$ does not contain 0 and by using a univariate interval Newton.

Two other contraction algorithms are often used in solvers. `HC4` [2] whose revise procedure traverses twice the tree representing the mathematical expression of the constraint for narrowing all the involved variable intervals. `3B` [11] or a variant `3BCID` [17] uses a shaving refutation principle similar to SAC [6].

## 3 Box-k Partial Consistency

As explained above, the Box-consistency yields an outer approximation/box of $1 \times 1$ subsystems $(c, x)$. The Box-k-consistency introduced in this paper generalizes Box-consistency by yielding an outer approximation of subsystems.

**Definition 4** *Let $P' = (X', C', B')$ be a subsystem of a numerical CSP $P = (X, C, B)$ ($|X'| = k$), in which the (output) variables in $X'$ are involved in at least one constraint in $C'$ and the* **input variables** *(i.e., the variables involved in at least one constraint in $C'$ which are not in $X'$) are replaced by their current interval in $B$.*

*The subsystem $P'$ is* **box-k-consistent** *if there exists a k-box of size 1 u.l.p. on every face of the k-box $B'$ for which all the constraints $c$ in $C'$ are "satisfied", i.e., $0 \in c(X')$.*

If a box-k-consistent subsystem has an empty set of input variables, note that this subsystem is also global hull consistent [5]. Thus, like for the standard box-consistency, the presence of input variables makes the box-k-consistency weaker than global hull consistency.

[1] The definition of box-consistency can be straightforwardly extended to inequalities.
[2] One Unit in the Last Place is the gap between two very close floating-point numbers.

Fig. 1-left shows an example of a $2 \times 2$ subsystem. The outer box is box-consistent since it optimally approximates the solution set of constraints $c_1$ and $c_2$ individually. The inner box is box-2-consistent since it optimally approximates the set of six "thick" solutions to both constraints. Constraints are thick because the input variables (e.g., $w_1$, $w_2$, $w_3$) are replaced by intervals.

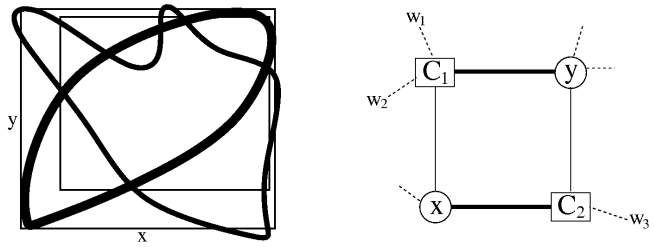


**Fig. 1.** Illustration of Box-2-consistency

Partial consistencies of NCSPs are generally defined modulo a precision $\epsilon$ that is used in practice by the corresponding algorithm to reach a fixpoint earlier. $\epsilon$ must then replace 1 u.l.p. in the previous definitions.

### 3.1 Benefits of Box-k-consistency

The following example theoretically shows that a contraction obtained by a $k \times k$ subsystem may be stronger than contraction on $1 \times 1$ subsystems and on the whole $n \times n$ system performed by an interval Newton. Consider the NCSP $P = (\{x, y, z\},\ \{x - y = 0,\ x + y + z = 0,\ (z - 1)(z - 4)(2x + y + 2) = 0\},\ \{[-10^6, 10^6], [-10^6, 10^6][-10, 10]\})$.

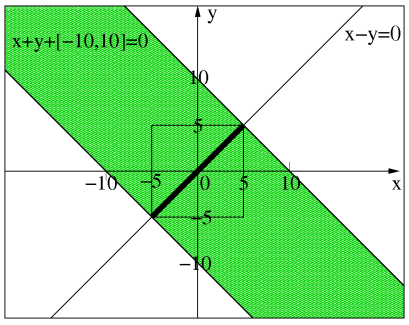


**Fig. 2.** Illustration of a subsystem of size 2, with $z = [-10, 10]$ as input variable. {[-5,5],[-5,5]} is box-2-consistent w.r.t. the 2 constraints $x - y = 0$ and $x + y + [z] = 0$.

Running `Box` and interval Newton on $P$ does not filter the box. Achieving Box-2-consistency on the $2 \times 2$ subsystem ({x,y}, {x-y=0, x+y+z=0}) narrows the intervals of $x$ and $y$ to $[-5, 5]$ as shown in Fig. 2. Also, if branching was used to find solutions, only two bisections (choice points) would be necessary to isolate the 3 solutions $\{(\frac{-2}{3}, \frac{-2}{3}, \frac{4}{3}), (-0.5, -0.5, 1), (-2, -2, 4)\}$. We should highlight that Newton on the whole system does not contract the box because it contains several solutions, whereas Newton on the $2 \times 2$ subsystem does because it contains only one (thick) solution (segment in bold). Of course, this small example is didactic. Experiments described in Section 6 show larger and non-linear instances highlighting the benefits of structural partial consistencies over stronger partial consistencies like 3B-consistency [11].

### 3.2 Achieving Box-k-consistency in well-constrained subsystems of equations

Enforcing Box-k-consistency in every subsystem of given size $k$ is too time-consuming and counter-productive in practice. The number of subsets of $k$ variables in a NCSP with $n$ variables is high and one needs to consider only promising subsystems.

We have thus used several criteria to reduce the number of subsystems that are candidate. We first select subsystems with only equations (no inequalities) because equations bring a great reduction of the search space and have nice properties. To understand these properties, we have to pay attention to NCSPs that admit a finite number of solutions. These NCSPs contains $n$ variables but also the same number $n$ of independent equations (additional inequalities can reduce the number of solutions). Also, the corresponding *bipartite constraint graph* verifies the following structural/graph property [1].

**Definition 5** *Let $P$ be a system of $n$ independent equations constraining $n$ variables. The vertices of the* **bipartite constraint graph** *$G$ corresponding to $P$ are the $n$ variables and the $n$ equations, and edges connect one equation to its involved variables.*

*The system of equations $P$ is* **(structurally) well-constrained** *if its constraint graph $G$ has a perfect matching [7].*

For instance, Fig. 1-right shows the perfect matching (bold-faced edges) of the corresponding subgraph. This structural well-constriction can be viewed as a necessary condition to obtain a finite set of solutions. It appears that interval Newton also requires this condition (while it is of course not sufficient) for contracting a box. Indeed, if the system is not structurally well-constrained, the jacobian matrix will necessarily be singular [1]. Our subsystems fulfill this condition because Interval Newton is used by our new Box-k-Revise procedure (see Section 4) to achieve faster a box-k-consistent subsystem. (Also, the time complexity of interval Newton is cubic in the number of variables, so that it is sometimes intractable to apply it to very large NCSPs. Instead, we could use interval Newton only inside subsystems.)

We finally require our subsystems be connected for performance considerations. Indeed, if a given subsystem of size $k$ contained several disconnected components of size at most $k'$ ($k' < k$), we could make it box-k-consistent by achieving box-k'-consistency in every component.

To sum up, restricting the subsystems to well-constrained and connected subgraphs of equations has two virtues. First, it allows a strong filtering in specific subparts of the system, which is useful for sparse NCSPs or for (globally) under-constrained ones, e.g., systems mixing equalities and inequalities. Second, it allows the use of an interval Newton to faster contract the subsystem.

## 4 Contraction Algorithm Using Well-constrained Subsystems as Global Constraints

Instead of contracting all the well-constrained subsystems of given size $k$, we have designed an AC3-like propagation that manages selected subsystems of different sizes: subsystems of size 1 but also well-constrained subsystems of larger size. Well-constrained subsystems are thus similar to *global constraints* [16, 10] that can be defined by the user or automatically (see Section 6).

All the subsystems are first put into a propagation queue and revised in sequence. When a variable domain is reduced more than a ratio $\rho_{propag}$, all the subsystems involving this variable are pushed into the queue, if they are not already in it. This propagation process is just specialized by the revise procedure used for contracting the subsystems of size greater than 1 and detailed below.

### 4.1 The `Box-k` revise procedure

The revise procedure is based on a branch & prune method limiting the bisection to the $k$ (output) variables $X$ of the subsystem, and using a *breadth-first* search. At the end of this local tree search, the current box is replaced by the hull of the leaves of the local tree. The algorithm `Box-k-Revise` is a generic procedure that achieves a box-k-consistent subsystem. The procedure manages a list $L$ of nodes that are leaves of the local tree. A leaf $l$ in $L$ has three significant components: $l.box$ designs the (n-dimensional) search space associated to the node; $l.precise$ is a boolean stating whether $l.box$ has reached the precision $\epsilon$ in all the dimensions ($\epsilon$ also yields the precision of the global solution); $l.certified$ is a boolean asserting whether $l.box$ contains a unique solution. The `box` parameter is the current global box (search space) when the revise procedure is called.

A combinatorial process (tree search) is performed by the `while` loop. At every iteration, one leaf in $L$, which is not *precise* and not *certified*, is selected, bisected and the two new sub-boxes are contracted. The search ends if all the leaves are tagged as *certified* or *precise* or if a limit $\tau_{leaves}$ in the number of leaves is reached. $\tau_{leaves}$ limits the memory storage requirement (see Section 4.5) and allows one to quickly propagate the obtained reductions to the other subsystems.

A leaf is simply selected in breadth-first order. We first tried a more sophisticated heuristic function for selecting a "large" box on the border of the hull of

**Algorithm 1** `Boxk-Revise` (**in-out** $L$, `box`; **in** $X$, $C$, $\epsilon$, `subContractor`, $\tau_{leaves}$, $\tau_{\rho_{io}}$)

```
UpdateLocalTree(L, box, X, C, ε, subContractor)
L' ← {l ∈ L s.t. ¬l.certified and ¬l.precise and ProcessLeaf?(l,X,C,τ_ρio)}
while 0 < L'.size and L.size < τ_leaves do
  l ← L'.front() /* Select a leaf in breadth-first order */
  (l1, l2) ←bisect(l, X)
  contract(l1, subContractor, X, C, ε)
  contract(l2, subContractor, X, C, ε)
  if l1.box ≠ ∅ then L.pushBack(l1) end if
  if l2.box ≠ ∅ then L.pushBack(l2) end if
  L.remove(l)
  L' ← {l ∈ L s.t. ¬l.certified and ¬l.precise and ProcessLeaf?(l,X,C,τ_ρio)}
end while
box ←hull(L) /* Outer approximation of the union of all the boxes l.box, l ∈ L */
```

the different leaves. The idea was to maximize the gain in volume on the current global box in case the selected leaf would be eliminated by filtering. This multi-dimensional generalization of the `BoxNarrow` algorithm (that shaves the bounds of the handled interval in the `Box` algorithm) has been discarded because it did not bring a significant gain in performance.

**Algorithm 2** `contract`(**in-out** $l$; **in** `subContractor`, $X$, $C$, $\epsilon$)

```
if ¬l.precise then
  if ¬l.certified then subContractor(l.box) end if
  if l.box ≠ ∅ and I-Newton(l.box,X) then l.certified ← true end if
  if maxDiameter(l.box) < ε then l.precise ← true end if
end if
```

The procedure `contract` is mainly parameterized by the contraction procedure `subContractor` (`HC4` [2] or `3BCID` [17] in our experiments). The scope $C$ of `subContractor` is the considered k-set of equations. After a call to `subContractor`, an interval Newton limited to the $k \times k$ subsystem is launched. If Newton certifies a unique solution in a leaf, `I-Newton` contracts $l.box$ and returns *true* so that this leaf is tagged as *certified*.

### 4.2 The `S-kB-Revise` variant

`S-kB-Revise` is the name of a variant of `Box-k-Revise` for which the entire system is used in the `contract` procedure. That is, the scope $C$ of `subContractor` includes the whole n-set of constraints, instead of the k-set of constraints attached to the subsystem. With `S-kB-Revise`, the k-set of constraints in the subsystem is just used by interval Newton. This variant brings additional filtering, but at a higher cost.

### 4.3 Reuse of the local tree (procedure `UpdateLocalTree`)

A simpler version of Algorithm 1 did not call the `UpdateLocalTree` procedure and simply initialized the list $L$ with the current box. However, instead of performing an intensive search effort in only one subsystem, we preferred to quickly propagate the obtained reductions to the other subsystems. Therefore the `UpdateLocalTree` procedure reuses the local tree (i.e., its leaves) that has been saved in a previous call to Algorithm 1. Every leaf in the current list $L$ is just updated by intersection with the current box and filtered with `subContractor`.

**Algorithm 3** `UpdateLocalTree` (**in-out** $L$; **in** `box`, $X$, $C$, $\epsilon$, `subContractor`)

```
if L = ∅ then
  L ← {Leaf(box)} /* Initialize the root of the local tree with the current box */
else
  for all l ∈ L do
    /* Update and contract every leaf of the stored local tree */
    if l.box ≠ (l.box ∩ box) then
      l.box ← l.box ∩ box
      contract(l, subContractor, C, ε))
      if l.box = ∅ then L.remove(l) end if
    end if
  end for
end if
```

In fact, the leaves of the local trees are also maintained in the global search tree. To do so, the list $L$ is implemented as a *backtrackable* data-structure updated in case of backtracking. It avoids redoing the same job in the subsystems several times, in particular when the *multisplit* splitting heuristic is chosen (see Section 5).

### 4.4 Lazy handling of a leaf (procedure `ProcessLeaf?`)

Our first experiments have shown us that handling a leaf in a local tree, i.e., bisecting it and contracting the two sub-boxes, was often counterproductive. We have then defined an input/output ratio $\rho_{io}$ that decides whether a given leaf of box $B$ must be handled in the local tree.

$$\rho_{io}(B, I, O, F) = \frac{Max_{x \in I}(smear(x))}{Max_{x \in O}(smear(x))}$$

The function `ProcessLeaf?` calculates $\rho_{io}$ in a leaf. If this ratio is larger than a threshold $\tau_{\rho_{io}}$, the leaf will not be handled in the current revise procedure.

$\rho_{io}$ is based on the well-known *smear* function [9] defined by:
$smear(x) := Max_{f \in F}(|\frac{\partial f}{\partial x}| \times Diam(x))$. This function is often used for selecting the next variable to be bisected in NCSPs (the variable with the largest smear evaluation).

The denominator of $\rho_{io}$ can be directly explained by it: output variables ($O$) with a great smear evaluation (implying a small ratio $\rho_{io}$) often lead to a great contraction when they are bisected inside the local subsystem tree. Desiring a small impact of the input variables ($I$) is less intuitive. We understand that large input domains generally lead to large output domains (i.e., leaf boxes) in the subsystem and thus yields a poor reduction. The same argument holds in fact for the derivatives of functions. To illustrate this point, let us take a subsystem of size 1 like $0.001\,y + x^2 - 1 = 0$ ($x$ is the output variable; $[x] = [y] = [-1, 1]$) having $\rho_{io} = \frac{0.002}{4} = 0.0005$. After one bisection on $x$, the subsystem contraction leads to a very small interval for $x$. A large interval would be obtained for $x$ if the considered subsystem was $y + x^2 - 1 = 0$ with $\rho_{io} = \frac{2}{4} = 0.5$.

### 4.5 Properties of the revise procedure

The following proposition formalizes the correctness, the memory and time complexities of the procedure `Box-k-Revise`.

**Proposition 1** *Let $P' = (X', C', B)$ be a subsystem of a CSP $P = (X, C, B)$, with $|X| = n$, $|C| = m$, $|X'| = |C'| = k$.*

*The procedure* `Box-k-Revise`*, called with $\tau_{leaves} = +\infty$ and $\tau_{\rho_{io}} = +\infty$, makes $P'$ box-k-consistent.*

*Let Diam be the largest interval diameter in $B$. Let $d$ be $log_2(\frac{Diam}{\epsilon})$, the maximum number of times a given interval must be bisected to reach the precision $\epsilon$.* [3]

*The memory complexity of* `Box-k-Revise` *is $O(k\,\tau_{leaves})$.*

*The number of calls to* `subContractor` *is $O(k\,d\,\tau_{leaves})$.*

*Proof.* The correction is based on the combinatorial process performed by the procedure `Box-k-Revise`. Called with $\tau_{leaves} = +\infty$ and with the subsystem made of $C'$, the procedure computes all the atomic boxes of precision $\epsilon$ in the subsystem before returning the hull of them, thus achieving roughly (i.e., assuming that the actual values of input variables are unknown) the global consistency of $P'$.

The memory complexity comes from the breadth-first search that must store the $O(\tau_{leaves})$ leaves of the local tree. The revise procedure works with n-dimensional boxes but, in order to save memory, stores at the end only $k$ intervals of a $k \times k$ subsystem.

The number of calls to `subContractor` is bounded by the number of nodes in the local search tree. The number of leaves of this tree is $\tau_{leaves}$ (corresponding to *living* boxes that can contain solutions) plus the number of *dead* leaves eliminated by filtering. For any living leaf $l$, the number of nodes created in the tree to reach $l$ is at most $2 \times d \times k$ since the root must be at most bisected $d$ times in all its $k$ dimensions. Although numerous such internal nodes are "shared" by several living leaves, this bounds the number of calls to a sub-filtering operator with $O(k\,d\,\tau_{leaves})$. □

[3] $d$ generally falls between 20 and 60 in NCSPs occurring in practice.

Another property allows us to better understand the gain in contraction obtained by the `S-kB-Revise` variant (see Section 4.2).

**Proposition 2** *Consider a propagation algorithm calling* `S-kB-Revise` *on all the subsystems of size $k$ in a given NCSP $P$.*
*This algorithm computes the $(k+2)B$-consistency of $P$.*

The kB-consistency, introduced by Lhomme [11], is a strong partial consistency related to the k-consistency (in finite-domain CSPs) restricted to the bounds of intervals. $3B$-consistency is similar to SAC-consistency [6]. It is known to be stronger (i.e., to better contract) than box-consistency (i.e., box-1-consistency). It appears that this result can be generalized to any $k > 1$.

## 5 Multidimensional Splitting

It turns out that the `Box-k-Revise` procedure has not only a contraction effect, but also provides a new way to make choice points, that is, to build the (global) search tree. This new splitting strategy is called *multidimensional splitting* (in short *multisplit*).

**Definition 6** *Consider a $k \times k$ subsystem $P'$ defined inside an NCSP $P = (X, C, B)$. Consider a set $S$ of $m$ boxes associated to $P'$ such that $S$ contains all the solutions to $P$, and the $m$ boxes obtained by projection on $P'$ of the boxes in $S$ are pairwise disjoint.*

*A* **multisplit** *of dimension $k$ consists in splitting the search space $B$ into the $m$ boxes in $S$.*

In practice, the $m$ boxes correspond to the leaves of a subsystem local tree. At the end of a `Box-k` propagation, our solving strategy makes a choice between a classical bisection and a multisplit. If all the subsystems have a ratio $\rho_m$ larger than a user-defined threshold $\tau_m$, then a standard bisection is performed. Otherwise, we multisplit the subsystem with the smallest ratio $\rho_m$, i.e., we replace the current box by the set $L$ of $m$ leaves associated to the local tree.

$$\rho_m = \frac{\sum_{l \in L} Volume(l)}{Volume(Hull(L))}$$

Multisplit generalizes a procedure used by `IBB` (see Section 6.1). `IBB` performs a multisplit once it finds the $m$ solutions (i.e., atomic boxes) in a given block. The difference here is that a multisplit may occur with *non* atomic boxes whose size has not reached the required precision.

## 6 Experiments

The `Box-k` based propagation algorithm has been implemented in the `Ibex` open source interval-based solver in `C++` [4, 3]. The variant with multisplit (`msplit`) performs a multisplit of a subsystem with the minimum ratio $\rho_m$, provided that $\rho_m < \tau_m$=0.99. All the competitors are also available in the same library, making the comparison fair.

### 6.1 Experiments on decomposed benchmarks

Ten *decomposed* benchmarks, described in [15, 14], appear in Table 1. They have been previously decomposed by equational algorithms (*eq*) like maximum-matching, or by more sophisticated geometrical algorithms (*geo*). They are challenging for general-purpose interval methods, but can efficiently be solved by `IBB` [15, 14].

#### Brief description of `IBB`

`IBB` is dedicated to decomposed systems, i.e., sparse systems of equations that have been first decomposed into a sequence of *irreducible* [1] well-constrained blocks/subsystems. Inter-Block Backtracking handles every block in the order provided by the sequence. It interleaves contraction steps (performed by `HC4` and interval Newton) and bisections inside the block until atomic boxes (solutions) are obtained. Choice points are then made: the variables of the block are replaced by one of the atomic boxes, i.e., they are considered constant in subsequent blocks.

We understand that the `Box-k-Revise` procedure plus multisplit represents a generalization of `IBB` in that the input variables domains of a subsystem are not necessarily atomic and that a multisplit is not necessarily performed after a subsystem handling. In other terms, the `IBB` block handling is not a revise procedure, it is just an ad-hoc procedure embedded in a dedicated algorithm. Applied to decomposed systems, the only information that our new approach does not exploit is the order between blocks which provides to `IBB` a useful splitting heuristic.

#### Experimental protocol

Every `Box-k` based strategy has been tuned with 6 different sets of parameter values: $\tau_{\rho_{io}}$ is 0.01, 0.2 or 0.8 (0.01 is always the best value on decomposed systems); the precision $\rho_{propag}$ used in the `HC4` propagation is 1% or 10%; All the other parameters have been empirically fixed: the precision $\rho_{propag}$ in the `Box-k` propagation is always 10%; the maximum number $\tau_{leaves}$ of leaves inside a subsystem tree is 10; the number of slices of `3BCID` in `Box-k(3BCID)` is 10. To be fair, the parameters of the competitor algorithms have been tuned so that 8 trials have been performed for `Box` and `HC4`, and 16 trials have been run for `3BCID`. For all the tests, the Newton ceil (size of maximum diameter under which interval Newton is run) is 10, and the same variable order is used in a round-robin strategy (except for `IBB` and for `Box-k` with multisplit).

The subsystems given to our `Box-k` propagation are defined automatically. The irreducible blocks produced by the `IBB` decomposition simply become the well-constrained subsystems handled by `Box-k-Revise`.

**Table 1.** Experimental results on IBB benchmarks. The first 3 columns include the name of the system, its number $n$ of variables and its number of solutions. The next three columns yield the CPU time (above) and the number of boxes, i.e., choice points (below), obtained on an Intel 6600 2.4 GHz by existing strategies based on HC4, Box or 3BCID followed by interval Newton (between two bisections selected in a round-robin way for the variable selection). The last four columns report the results obtained by our algorithms on the same computer: Box-k-Revise parameterized by subContractor=HC4 or subContractor=3BCID, with multisplit (msplit) or without. To be the closest to IBB, Box-k-Revise, and not the S-kB-Revise variant, is used by our constraint propagation algorithm.

| **Benchmark** | $n$ | #sols | **HC4** | **Box** | **3BCID** | **IBB** | **Box-k(HC4)** | | **Box-k(3BCID)** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | **msplit** | | **msplit** |
| **Chair(eq)** | 178 | 8 | >3600 | >3600 | >3600 | **0.27** | >3600 | 16.5 | >3600 | **0.52** |
| 1x15,1x13,1x9,5x8,3x6,... | | | | | | | | 575 | | 15 |
| **Latham(eq)** | 102 | 96 | >3600 | >3600 | 39.9 | **0.17** | **0.94** | 1.35 | 1.5 | 1.08 |
| 1x13,1x10,1x4,25x2,25x1 | | | | | 587 | | 839 | 199 | 991 | 189 |
| **Ponts(eq)** | 30 | 128 | 33.4 | 33.4 | 1.89 | **0.59** | 6.85 | 8.19 | 0.79 | **0.71** |
| 1x14,6x2,4x1 | | | 20399 | 20399 | 357 | | 783 | 231 | 307 | 231 |
| **Ponts(geo)** | 38 | 128 | 44.1 | 44.1 | 2.6 | **0.16** | 2.01 | 0.31 | 1.45 | **0.39** |
| 13x2,12x1 | | | 18363 | 18363 | 685 | | 6711 | 767 | 6711 | 767 |
| **Sierp3(geo)** | 124 | 198 | >3600 | >3600 | 77.5 | **0.62** | 49.0 | **1.38** | 52.5 | 1.77 |
| 44x2,36x1 | | | | | 1727 | | 84169 | 1513 | 84169 | 1513 |
| **Star(eq)** | 46 | 128 | >3600 | >3600 | 4.9 | **0.05** | 35.6 | **0.12** | 44.0 | 0.26 |
| 3x6,3x4,8x2 | | | | | 283 | | 44195 | 263 | 44023 | 263 |
| **Tangent(eq)** | 28 | 128 | 77 | 77 | 2.1 | **0.08** | 1.74 | **0.08** | 1.87 | 0.14 |
| 1x4,10x2,4x1 | | | 390903 | 390903 | 753 | | 12027 | 255 | 12235 | 255 |
| **Tangent(geo)** | 42 | 128 | – | – | 7.38 | **0.08** | 0.80 | 0.19 | 0.80 | **0.19** |
| 2x4,11x2,12x1 | | | | | 859 | | 1415 | 251 | 1407 | 251 |
| **Tetra(eq)** | 30 | 256 | 1281 | 1281 | 12.3 | **0.63** | 33.6 | 1.06 | 13.57 | **0.76** |
| 1x9,4x3,1x2,7x1 | | | 607389 | 607389 | 1713 | | 4619 | 483 | 2243 | 483 |
| **Sierp3(eq)** | see Section 6.2 | | | | | >5000 | see Section 6.2 | | | |

### Results

Strategies based on HC4, Box and 3BCID followed by interval Newton are not competitive at all with Box-k and IBB on the tested decomposed systems. The comparison of Box-k against IBB is very positive because the CPU times reported for IBB are really the best that have never been obtained with any variant of this dedicated algorithm. Also, no timeout is reached by Box-k+multisplit and IBB is on average only twice faster than Box-k(3BCID) (at most 6 on Latham). As expected, the results confirm that multisplit is always relevant for decomposed benchmarks. For the benchmark Sierp3(eq) (the fractal Sierpinski at level 3 handled by an equational decomposition), an equational decomposition makes appear a large irreducible $50 \times 50$ block of distance constraints. This renders IBB unefficient on it (timeout).

### 6.2 Experiments on structured systems

Eight *structured* systems appear in Table 2. They are scalable chains of constraints of reasonable arity [12]. They are denoted *structured* because they are not sufficiently sparse to be decomposed by an equational decomposition, i.e.,

the system contains only one irreducible block, thus making `IBB` pointless. A brief and manual analysis of the constraint graph of every benchmark has led us to define a few well-constrained subsystems of reasonable size (between 2 and 10). In the same way, we have replaced the $50 \times 50$ block in `Sierp3(eq)` by $6 \times 6$ and $2 \times 2$ `Box-k` subsystems.

**Table 2.** Results on structured benchmarks. The same protocol as above has been followed, except that the solving strategy is more sophisticated. Between two bisections, the propagation with subsystems follows a `3BCID` contraction and an interval Newton. The four `Box-k` columns report the results obtained by the `S-kB-Revise` variant. The results obtained by `Box-k-Revise` are generally worse and appear, with multisplit only, in the last two columns.

| **Benchmark** | $n$ | #sols | **HC4** | **Box** | **3BCID** | **Box-k(HC4)** | | **Box-k(3BCID)** | | **Box-k-Revise** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | **msplit** | | **msplit** | HC4 | 3BCID |
| **Bratu** | 60 | 2 | 58 | 626 | 48.7 | 47.0 | **33.0** | 135 | 126 | 86.4 | 96.2 |
| 29x3 | | | 15653 | 13707 | 79 | 39 | 17 | 43 | 25 | 125 | 129 |
| **Brent** | 10 | 1015 | 1383 | 127 | **17.0** | 28.5 | 20.2 | 44.9 | 31.0 | 20.8 | 34.9 |
| 2x5 | | | 7285095 | 42191 | 9849 | 2975 | 4444 | 4585 | 1309 | 5215 | 4969 |
| **BroydenBand** | 20 | 1 | >3600 | 0.17 | **0.11** | 0.45 | 0.15 | 0.91 | 0.31 | 0.30 | 0.28 |
| 1x6,3x5 | | | | 1 | 21 | 4 | 19 | 17 | 3 | 7 | 3 |
| **BroydenTri** | 30 | 2 | 1765 | **0.16** | 0.25 | 0.22 | 0.24 | 0.39 | 0.29 | 0.19 | 0.23 |
| 6x5 | | | 42860473 | 63 | 25 | 11 | 19 | 9 | 3 | 19 | 17 |
| **Reactors** | 30 | 120 | >3600 | >3600 | 288 | 340 | 315 | 81.4 | **67.5** | 250 | 194 |
| 3x10 | | | | | 39253 | 14576 | 10247 | 1038 | 788 | 35867 | 21465 |
| **Reactors2** | 10 | 24 | >3600 | >3600 | 28.8 | **9.5** | 12.3 | 10.4 | 12.2 | 9.93 | 11.9 |
| 2x5 | | | | | 128359 | 4908 | 10850 | 4344 | 5802 | 5597 | 5353 |
| **Sierp3Bis(eq)** | 83 | 6 | >3600 | >3600 | 4917 | >3600 | >3600 | >3600 | **389** | >3600 | 4503 |
| 1x14,6x6,15x2,3x1 | | | | | 44803 | | | | 218 | | 122409 |
| **Trigexp1** | 30 | 1 | >3600 | 13 | **0.08** | **0.08** | **0.08** | **0.08** | 0.09 | 0.08 | 0.08 |
| 6x5 | | | | 27 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **Trigexp2** | 11 | 0 | 1554 | >3600 | 83.7 | 81.2 | 85.7 | 105 | 83.0 | **80.6** | 82.1 |
| 2x4,2x3 | | | 2116259 | | 16687 | 15771 | 16755 | 3797 | 2379 | 15771 | 11795 |

Standard strategies based on `HC4` or `Box` followed by interval Newton are generally not competitive with `Bok-k` on the tested benchmarks. The solving strategy based on `S-kB-Revise` with `subContractor=3BCID` (column `Box-k(3BCID)`) appears to be a robust hybrid algorithm that is never far behind `3BCID` and is sometimes clearly better. The gain w.r.t. `3BCID` falls indeed between 0.7 and 12. The small number of boxes highlights the additional filtering power brought by well-constrained subsystems. Again, multisplit is often the best option.

The success of `Box-k` on `Sierp3Bis(eq)` has led us to try a particular version of `IBB` in which the inter-block filtering [15] is performed by `3BCID`. Although this variant seldom shows a good performance, it can solve `Sierp3(eq)` in 330 seconds.

### 6.3 Benefits of sophisticated features

Tables 3 has finally been added to show the individual benefits brought by two features: the user parameter $\tau_{\rho_{io}}$ driving the procedure `ProcessLeaf?` and the backtrackable list of leaves used to reuse the job achieved inside the subsystems.

**Table 3.** Benefits of the backtrackable data structure (BT) and of $\tau_{\rho_{io}}$ in the `Box-k`-based strategy. Setting $\tau_{\rho_{io}} = \infty$ means that subsystem leaves will be always processed in the revise procedure.

| | **Chair** | **Latham** | **Ponts(eq)** | **Ponts(geo)** | **Sierp3(geo)** | **Star** | **Tan(eq)** | **Tan(geo)** | **Tetra** |
|---|---|---|---|---|---|---|---|---|---|
| BT, $\tau_{\rho_{io}}$ | 0.52 | 1.08 | 0.71 | 0.31 | 1.38 | 0.12 | 0.08 | 0.19 | 0.76 |
| ¬ BT, $\tau_{\rho_{io}}$ | 10.8 | 4.61 | 1.51 | 1.27 | 23.9 | 2.34 | 0.71 | 1.58 | 2.13 |
| BT, $\tau_{\rho_{io}} = \infty$ | 23.4 | 4.71 | 2.60 | 1.00 | 23.8 | 1.67 | 1.09 | 1.81 | 3.57 |
| ¬ BT, $\tau_{\rho_{io}} = \infty$ | 24.2 | 6.60 | 2.80 | 1.11 | 23.9 | 2.40 | 1.15 | 1.82 | 3.54 |

| | **Bratu** | **Brent** | **BroyB.** | **BroyT.** | **Sierp3B(eq)** | **Reac.** | **Reac.2** | **Trigexp1** | **Trigexp2** |
|---|---|---|---|---|---|---|---|---|---|
| BT, $\tau_{\rho_{io}}$ | 33.0 | 20.2 | 0.15 | 0.24 | 389 | 67 | 12.2 | 0.08 | 83 |
| ¬ BT, $\tau_{\rho_{io}}$ | 33.2 | 21.0 | 0.14 | 0.23 | 411 | 97 | 12.0 | 0.07 | 85 |
| BT, $\tau_{\rho_{io}} = \infty$ | 33.9 | 23.8 | 0.38 | 0.28 | 519 | 164 | 13.1 | 0.10 | 103 |
| ¬ BT, $\tau_{\rho_{io}} = \infty$ | 33.0 | 28.7 | 0.40 | 0.38 | 533 | 401 | 18.7 | 0.07 | 148 |

Every cell reports the best result (CPU time in second) among both subcontractors. Multisplit is allowed in all the tests. The first line of results corresponds to the implemented and sophisticated revise procedure; the next ones correspond to simpler versions for which at least one of the two advanced features has been removed.

Three main observations can be drawn. First, when a significant gain is brought by the features on a given system, then this system is efficiently handled against competitors in Tables 1 and 2. Second, $\tau_{\rho_{io}}$ seems to have a better impact on performance than the backtrackable list, but the difference is slight. Third, several systems are only slightly improved by one of both features, whereas the gain is significant when both are added together. This is true for most of the `IBB` benchmarks. On these systems, between 2 bisections in the search tree, it often occurs that a job inside *several* subsystems leads to identify atomic boxes (some others are not fully explored thanks to $\tau_{\rho_{io}}$). Although we multisplit only one of these subsystems, the job on the others is saved in the backtrackable list.

## 7 Conclusion

We have proposed a new type of filtering algorithms handling $k \times k$ well-constrained subsystems in an NCSP. $k \times k$ interval Newton calls and selected bisections inside such subsystems are useful to better contract decomposed and structured NCSPs. In addition, the local trees built inside subsystems allow a solving strategy to learn choice points bisecting several variable domains simultaneously.

Solving strategies based on `Box-k` propagations and multisplit have mainly three parameters: the choice between `Box-k-Revise` and `S-kB-Revise` (although `Box-k-Revise` seems better suited only for decomposed systems), the choice of sub-contractor (although `3BCID` seems to be often a good choice), and $\tau\rho_{io}$. This last parameter appears to be finally the most important one.

On decomposed and structured systems, our first experiments suggest that our new solving strategies are more efficient than standard general-purpose strategies based on `HC4`, `Box` or `3BCID` (with interval Newton). `Box-k`+multisplit

can be viewed as a generalization of `IBB`. It can also solve large decomposed NCSPs with relatively small blocks in less than one second, but can also handle structured NCSPs that `IBB` cannot treat.

Subsystems have been automatically added in the decomposed systems, but have been manually added in the structured ones, as global constraints. In this paper, we have validated the fact that handling subsystems could bring additional contraction and relevant multi-dimensional choice points. The next step is to automatically select a relevant set of subsystems. We believe that an adaptation of maximum-matching machinery or other graph-based algorithms along with a criterion similar to $\rho_{io}$ could lead to efficient heuristics.

## Acknowledgments

We thank the referees for their helpful comments.

## References

1. S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of Constraint Systems. In *Compugraphic*, 1993.
2. F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.
3. G. Chabert. `www.ibex-lib.org`, 2009.
4. G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
5. J. Cruz and P. Barahona. Global Hull Consistency with Local Search for Continuous Constraint Solving. In *Proc. EPIA, LNAI 2258*, pages 349–362, 2001.
6. R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.
7. A.L. Dulmage and N.S. Mendelsohn. Covering of Bipartite Graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.
8. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.
9. R.B. Kearfott and M. Novoa III. INTBIS, a portable interval Newton/Bisection package. *ACM Trans. on Mathematical Software*, 16(2):152–157, 1990.
10. Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J.P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.
11. O. Lhomme. Consistency Tech. for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.
12. J-P. Merlet. `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`, 2009.
13. A. Neumaier. *Int. Meth. for Systems of Equations*. Cambridge Univ. Press, 1990.
14. B. Neveu, G. Chabert, and G. Trombettoni. When Interval Analysis helps Interblock Backtracking. In *Proc. CP, LNCS 4204*, pages 390–405, 2006.
15. B. Neveu, C. Jermann, and G. Trombettoni. Inter-Block Backtracking: Exploiting the Structure in Continuous CSPs. In *Selected papers WS COCOS, LNCS 3478*, pages 15–30, 2005.
16. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. AAAI'94*, pages 362–367, 1994.
17. G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP, LNCS 4741*, pages 635–650, 2007.