Box Consistency through Adaptive Shaving

Alexandre Goldsztejn
CNRS, LINA, UMR 6241
2 rue de la Houssinière
BP 92208, F-44000 NANTES
Alexandre.Goldsztejn@univ-nantes.fr

Frédéric Goualard
Université de Nantes
Nantes Atlantique Université
CNRS, LINA, UMR 6241
2 rue de la Houssinière
BP 92208, F-44000 NANTES
Frederic.Goualard@univ-nantes.fr

ABSTRACT

The canonical algorithm to enforce box consistency over a constraint relies on a dichotomic process to isolate the leftmost and rightmost solutions. We identify some weaknesses of the standard implementations of this approach and review the existing body of work to tackle them; we then present an adaptive shaving process to achieve box consistency by tightening a domain from both bounds inward. Experimental results show a significant improvement over existing approaches in terms of robustness for difficult problems.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General—*interval arithmetic*

General Terms

Algorithms, Performance

Keywords

Local consistency, constraint programming, Newton method

1. INTRODUCTION

Box consistency [1] is a local consistency notion for continuous constraints. Stronger consistencies such as *bounds consistency* build on box consistency to achieve better domain narrowing, which makes it pivotal in solving difficult constraint systems. Box consistency is usually enforced on unary projections of constraints by a combination of binary search and interval Newton steps to isolate leftmost and rightmost "quasi-zeroes" in the domain of a variable.

The canonical algorithm to enforce box consistency, described in details by Van Hentenryck *et al.* [6], is quite robust and efficient on many problems. However, as we will see in Section 2, it is an intrinsically optimistic algorithm that always attempts to reduce large subdomains first during the dichotomic process without taking into account information on the successes of its previous attempts. As experimental evidences presented in Section 4 show, this behavior makes it a largely suboptimal algorithm for many problems.

Early on, McAllester *et al.* [4] proposed a different view of the original algorithm to enforce box consistency in which it is no longer described as a recursive dichotomic process but as some kind of shaving process where a strict enforcement of box consistency is no longer the ultimate goal. In practice, this approach works well, except for those difficult problems for which it pays off to spend more time narrowing domains further in the contracting operator and defer less to the binary search process of the solver.

Following this track, we present in Section 3 an algorithm that enforces box consistency by an adaptive shaving process that takes into account past difficulties encountered in narrowing a domain down. Experimental evidences reported in Section 4 show that our new algorithm has more regular performances than McAllester *et al.*'s algorithm, even more so for difficult problems, where the later may flounder.

2. BOX CONSISTENCY

Interval Constraint Programming makes cooperate *contracting operators* to prune the domains of variables (intervals with floating-point bounds from the set $\mathbb{I}$) with smart *propagation algorithms* to ensure consistency among all the constraints. *Exploration algorithms* recursively split domains until a desired precision is achieved.

The amount of pruning obtained from one constraint is controlled by the level of consistency enforced. Box consistency considers interval extensions of the constraints. It is therefore parameterized by the interval extension used. In this paper, we only consider the *natural interval extension* of constraints, though our approach is not limited in any way to it. The same holds for the kind of constraints considered: while we limit ourselves to equations for the sake of simplicity, box consistency—and the algorithms presented—may be applied to inequations as well.

With these restrictions in mind, the definition of box consistency is as follows (using the notations for interval arithmetic advocated by Kearfott *et al.* [3], where interval quantities are boldfaced):

Definition 1 (Box consistency). The constraint $f(x_1,\dots,x_n)=0$ is box consistent with respect to a variable x_i and a box of domains $\boldsymbol{B}=\boldsymbol{I_1}\times\cdots\times\boldsymbol{I_n}$ if and only if:

$$\begin{cases} 0\in\boldsymbol{f}(\boldsymbol{I_1},\dots,\boldsymbol{I_{i-1}},[\underline{I_i},\underline{I_i}^{+}],\boldsymbol{I_{i+1}},\dots,\boldsymbol{I_n}) \\ \text{and} \\ 0\in\boldsymbol{f}(\boldsymbol{I_1},\dots,\boldsymbol{I_{i-1}},[\overline{I_i}^{-},\overline{I_i}],\boldsymbol{I_{i+1}},\dots,\boldsymbol{I_n}), \end{cases} \tag{1}$$

Algorithm 1 Computing a box consistent interval

```
[$\boldsymbol{bc3revise(lnar,rnar)}$] in: $\boldsymbol{g}\colon \mathbb{I}\to\mathbb{I}$; in: $\boldsymbol{I}\in\mathbb{I}$
# Returns the largest interval included in $\boldsymbol{I}$ that is
#  box consistent with respect to $g$. The algorithm is
# parameterized by the "lnar" and "rnar" methods used.
begin
    1  if $0\notin \boldsymbol{g}(\boldsymbol{I})$:
    2      return $\varnothing$
    3  return $lnar(\boldsymbol{g}, rnar(\boldsymbol{g},\boldsymbol{I}))$
end
```

where $\boldsymbol{I}_k = [\underline{I_k}, \overline{I_k}]$ are intervals with floating-point bounds, a^+ (resp., a^-) is the smallest floating-point number greater than (resp., the largest floating-point number smaller than) the floating-point number a, and $\boldsymbol{f}$ is the natural interval extension of f.

A constraint $f(x_1,\dots,x_n)=0$ is box consistent with respect to $\boldsymbol{B}$ if it is box consistent with respect to $x_1,\dots,x_n$ and $\boldsymbol{B}$. A constraint system is box consistent if all constraints are box consistent.

In order not to lose any potential solution, an algorithm that enforces box consistency of a constraint with respect to a variable x_i and a box of domains must return the largest domain $\boldsymbol{I_i'}\subseteq\boldsymbol{I_i}$ that verifies Eq. (1).

Given a real function $f\colon\mathbb{R}^n\to\mathbb{R}$ and a box of domains $\boldsymbol{B}=\boldsymbol{I_1}\times\cdots\times\boldsymbol{I_n}\in\mathbb{I}^n$, we are hereafter only concerned with interval projections $\boldsymbol{g}\colon\mathbb{I}\to\mathbb{I}$ of f with respect to $\boldsymbol{B}$, of the form:

$$\boldsymbol{g}(x)=\boldsymbol{f}(\boldsymbol{I_1},\dots,\boldsymbol{I_{i-1}},x,\boldsymbol{I_{i+1}},\dots,\boldsymbol{I_n}),\quad i\in\{1,\dots n\}.$$

Algorithm *bc3revise*, presented by Alg. 1, enforces box consistency by searching the leftmost and rightmost *canonical domains*[1] in the domain $\boldsymbol{I}$ of x for which $\boldsymbol{g}$ evaluates to an interval containing 0. It is parameterized by the two procedures *lnar* and *rnar* that perform the search for the left and right bounds respectively.

Algorithm *bc3revise* first tries to move the right bound of $\boldsymbol{I}$ to the left, and then proceeds to move its left bound to the right (this order could be reversed with no impact on performances, though).

In the original algorithm [1, 6], the search within *lnar* and *rnar* is performed by a dichotomic search aided with Newton steps to accelerate the process. Algorithm 2 describes the search of a quasi-zero to update the left side of $\boldsymbol{I}$. The procedure $rnar_{\text{vhmak}}$ to update the right bound works along the same lines and is, therefore, omitted.

The Newton procedure in Alg. 2 computes a step of the *Interval Newton algorithm*, that is, at iteration $j+1$:

$$\boldsymbol{I^{(j+1)}}\leftarrow\boldsymbol{I^{(j)}}\cap\left(\kappa(\boldsymbol{I^{(j)}})-\frac{\boldsymbol{g}(\kappa(\boldsymbol{I^{(j)}}))}{\boldsymbol{g'}(\boldsymbol{I^{(j)}})}\right)\tag{2}$$

with $\kappa(\boldsymbol{I})$ being a floating point number in $\boldsymbol{I}$. The notation *Newton*$^{(\star)}$ denotes the computation of Newton steps until reaching a fixpoint; we also use later the notation *Newton*$^{(k)}$ to denote an unspecified number of Newton steps (determined by the context). The "*hull*" notation $\Box(\mathcal{S})$ denotes the smallest interval (in the sense of set inclusion) containing the set $\mathcal{S}$; the notation $\mathsf{m}(\boldsymbol{I})$ denotes the midpoint of the interval $\boldsymbol{I}$.

[1]A non-empty interval $[a,b]$ is canonical if $a^+\geqslant b$.

Algorithm 2 Computing a box consistent left bound

```
[$\boldsymbol{lnar_{\text{vhmak}}}$] in: $\boldsymbol{g}\colon\mathbb{I}\to\mathbb{I}$; in: $\boldsymbol{I}\in\mathbb{I}$
# Returns an interval included in $\boldsymbol{I}$
# with the smallest left bound $l$ such that $0\in\boldsymbol{g}([l,l^+])$
begin
    1  $r\leftarrow\overline{I}$ # Preserving the right bound
    2  if $0\notin\boldsymbol{g}(\boldsymbol{I})$:  # No solution in $\boldsymbol{I}$
    3      return $\varnothing$
    4  $\boldsymbol{I}\leftarrow$ Newton$^{(\star)}(\boldsymbol{g},\boldsymbol{g'},\boldsymbol{I})$ # Interval Newton steps
    5  if $\boldsymbol{I}=\varnothing$:
    6      return $\varnothing$
    7  if $0\in\boldsymbol{g}([\underline{I},\underline{I}^+])$: # Left bound is box consistent?
    8      return $[\underline{I},r]$
    9  else:
   10      $(\boldsymbol{I_1},\boldsymbol{I_2})\leftarrow$ split$(\boldsymbol{I})$ # $\boldsymbol{I_1}\leftarrow[\underline{I},\mathsf{m}(\boldsymbol{I})]$, $\boldsymbol{I_2}\leftarrow[\mathsf{m}(\boldsymbol{I}),\overline{I}]$
   11      $\boldsymbol{I}\leftarrow$ lnar$_{\text{vhmak}}(\boldsymbol{g},\boldsymbol{I_1})$
   12      if $\boldsymbol{I}=\varnothing$:
   13          return $\Box\left(\text{lnar}_{\text{vhmak}}(\boldsymbol{g},\boldsymbol{I_2})\cup\{r\}\right)$
   14      else:
   15          return $\Box\left(\boldsymbol{I}\cup\{r\}\right)$
end
```

Geometrically, an Interval Newton step reduces a domain $\boldsymbol{I}$ for an interval function $\boldsymbol{g}$ by computing the intersection between the x-axis and the area that contains all the lines passing through points in the segment $(\kappa(\boldsymbol{I}),\underline{\boldsymbol{g}(\kappa(\boldsymbol{I}))})$–$(\kappa(\boldsymbol{I}),\overline{\boldsymbol{g}(\kappa(\boldsymbol{I}))})$ with slopes ranging in $\boldsymbol{g'}(\boldsymbol{I})$ (see Fig. 1 for an illustration). By the Mean Value Theorem, this area always encloses the graph of $\boldsymbol{g}$ on $\boldsymbol{I}$, and thence its intersection with the x-axis contains all the solutions.

The Newton step expression (2) uses an extended version of the division (hereafter noted "$\oslash$") to return a union of two open-ended intervals whenever $\boldsymbol{g'}(\boldsymbol{I^{(j)}})$ contains 0. The subtraction and the intersection operators are modified accordingly: The intersection operator is applied to an interval $(\boldsymbol{I^{(j)}})$ and a union of two intervals (result of the subtraction), and returns an interval.

The $lnar_{\text{vhmak}}$ and $rnar_{\text{vhmak}}$ methods of the original algorithm use the midpoint $\mathsf{m}(\boldsymbol{I^{(j)}})$ of $\boldsymbol{I^{(j)}}$ for $\kappa(\boldsymbol{I^{(j)}})$. Benhamou *et al.* [1] considered using the left bound for *lnar* and the right bound for *rnar* but found that it could lead to slow convergence phenomena. On the other hand, as noted by McAllester *et al.* [4], a Newton expansion on a bound always yields some reduction of the domain.

For example, Fig. 2(a) presents a situation where the Newton method cannot tighten a domain when expanding on the center of the domain (green area), while expanding on the left bound allows to tighten the domain on the left significantly (red area). However, Fig. 2(b) shows that this ability to always achieve some reduction may be more a curse than a blessing for problems where the derivative varies widely on the domain considered: no reduction occurs on the original domain $[-8,10]$, leading to splitting it and investigating separately the left part $[-8,1]$ (red line below the x-axis); from there, each step of the Newton method with an expansion on the left bound is able to shave the domain ever so slightly, leading to a very slow convergence (the reductions due to Newton steps are shown by the dashed lines below the x-axis). On the other hand, expanding on the center of the domain exhibits a different, more efficient behavior: no reduction occurs on the original domain, as in the previous case; the left half sub-domain $[-8,1]$ (lowest red line above the x-axis) is investigated; no reduction occurs be-

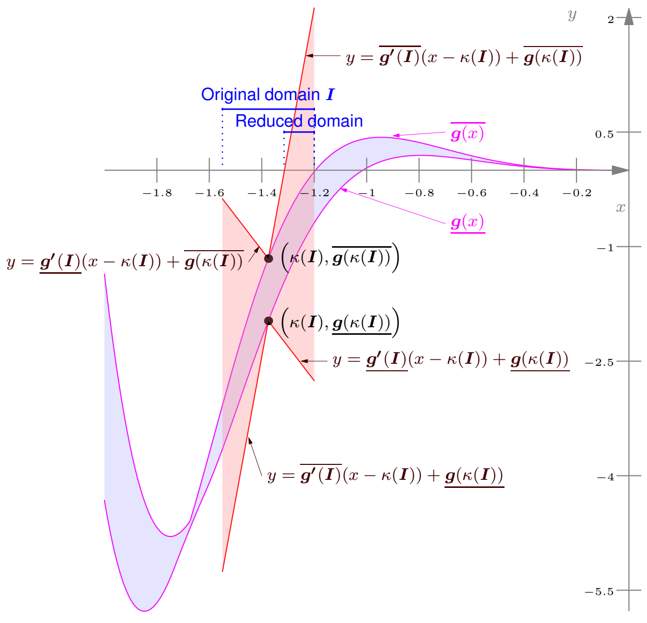


Figure 1: Interval Newton step with $I = [-1.55, -1.2]$ and $\kappa(\boldsymbol{I}) = \mathsf{m}(\boldsymbol{I})$

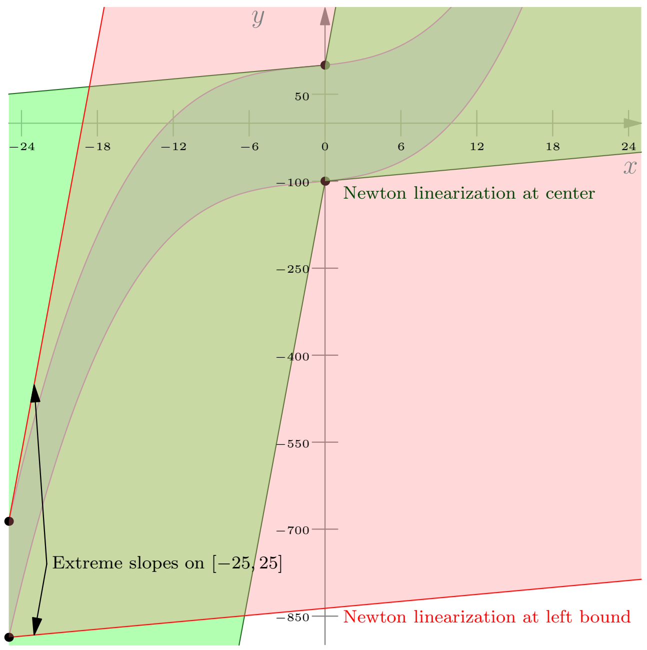
cause the smallest and largest slopes corresponding to the derivative of the function on $[-8, 1]$ are so steep that only a small portion in the middle of $[-8, 1]$ could be removed; since we restrict ourselves to intervals, the whole domain is kept unchanged. The algorithm then splits the domain and considers $[-8, -3.5]$, which can be discarded immediately by a simple evaluation of the function on it; the right part $[-3.5, 1]$ is then considered, and so on. Note that, on this example, the algorithm with an expansion on the center exhibits a quite inefficient behavior of its own in that Newton steps are performed but can never achieve any reduction due to the steepness of the slopes on the $[0, 1]$ subdomain.

3. BOX CONSISTENCY BY SHAVING

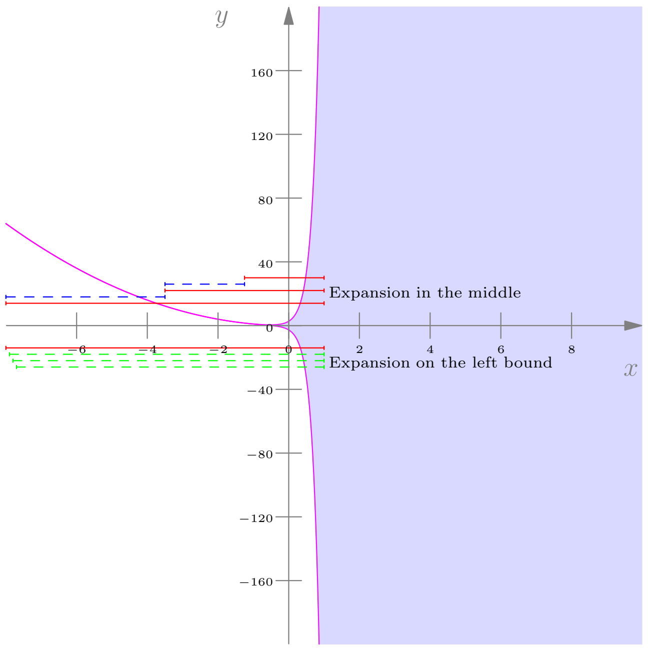
McAllester *et al.* [4] consider the enforcement of box consistency as a shaving process where the bounds of a domain are moved inward by trying to discard "slices" of varying sizes on the left and right of the domain. The result is an algorithm whose most natural presentation is no longer recursive (*cf.* Alg. 2). We present in Alg. 3 a generic shaving algorithm for the left bound that uses the interval Newton operator[2]. The slice to be processed is defined by choosing a value p (Line 4). The reduction process is then applied to the slice **guess** $= [\underline{I}, p]$.

To shave a domain $[a, b]$ on, say, the left side, we must choose a slice $[a, p] \subseteq [a, b]$ that we will try to discard, and a value for $\kappa([a, p])$. The larger $[a, p]$ the better, provided we are able to discard it. On the other hand, a large $[a, p]$

[2]Note, however, that McAllester *et al.*'s algorithm slightly differs from Alg. 3 in that their purpose is only to move the left and right bounds inward by a "reasonable" amount (in their paper, they consider that reducing a domain by 10% or more is a reasonable goal to achieve for a contracting operator), which means that they may not enforce box consistency.

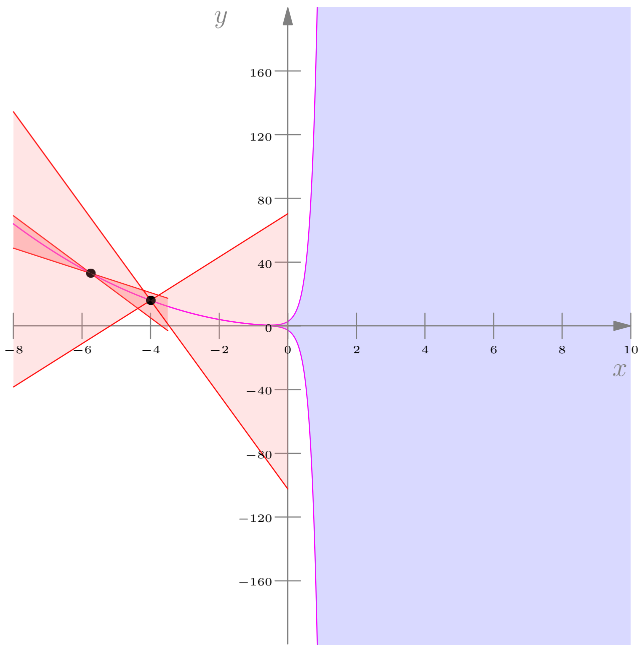


(a) Expansion on bound vs. expansion on center

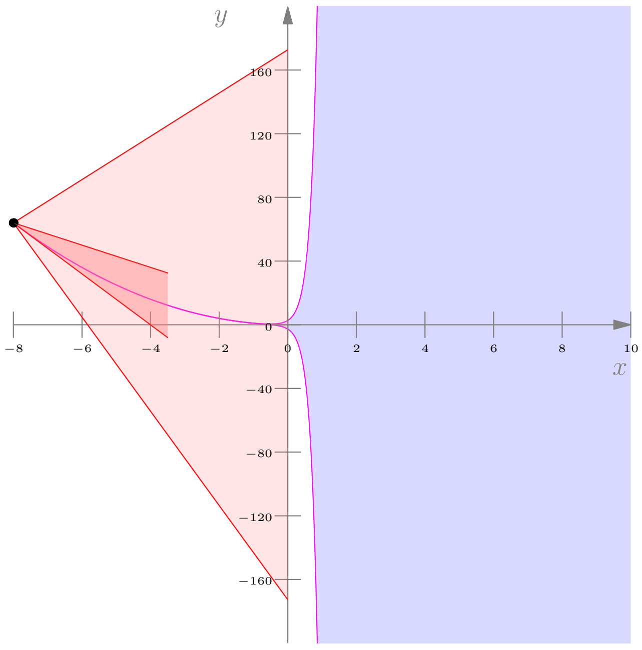


(b) Slow convergence

Figure 2: Box consistency enforcement. Interval functions have magenta borders and a light blue surface. In Figure 2(b), red domains are obtained by dichotomy; blue domains are discarded by an evaluation of the function; green domains are narrowed by a Newton step.



(a) Expansion on center



(b) Expansion on left bound

Figure 3: Shaving process for the left side. Initial domain is $[-8, 10]$; Slices considered are $[-8, -3.5]$ (dark red area) and $[-8, 0]$ (light red area).

Algorithm 3 Enforcing box consistency by shaving

```
[$\mathit{lnar}_{\mathbf{shaving}}$] in: $\boldsymbol{g}\colon \mathbb{I}\to\mathbb{I}$; in: $\boldsymbol{I}\in\mathbb{I}$
begin
    # Non-empty $\boldsymbol{I}$ is not box consistent on left?
    while $\boldsymbol{I}\neq\varnothing \wedge 0\notin \boldsymbol{g}([\underline{I},\underline{I}^{+}])$:
        $\boldsymbol{I}\leftarrow[\underline{I}^{+},\overline{I}]$ # We already know $0\notin[\underline{I},\underline{I}^{+}]$
        $p\leftarrow$ choose a floating-point in $\boldsymbol{I}$
        $\mathbf{guess}\leftarrow[\underline{I},p]$
        if $0\notin\boldsymbol{g}(\mathbf{guess})$:
            $\mathbf{reduced}\leftarrow\varnothing$
        else:
            # Performing one or more Newton steps
            # (determined by the instantiation of the alg.)
            $\mathbf{reduced}\leftarrow \mathit{Newton}^{(k)}(\boldsymbol{g},\boldsymbol{g}',\mathbf{guess})$
        # Keep the right bound unchanged:
        $\boldsymbol{I}\leftarrow\Box\,(\mathbf{reduced}\cup(\boldsymbol{I}\setminus\mathbf{guess}))$
    return $\boldsymbol{I}$
end
```

often entails a large domain for the derivative on $[a, p]$, which leads to less shaving. The same problem holds for the choice of $\kappa([a, p])$: taking $\kappa([a, p]) = a$ ensures that we will shave the left part of $[a, b]$, however slightly. Values farther from a potentially offer better cuts, while they may also lead to no reduction at all.

All these situations are presented in Figures 3(a) and 3(b): Given the initial domain $[-8, 10]$, choosing a small slice, such as $[-8, -3.5]$, is often a sure-fire way of achieving some reduction, which may be large compared to the size of the slice, though it is proportionally small compared to the initial domain. Here, we obtain approximately $[-3.68, -3.5]$ for the expansion point on the center (dark red cone in Fig. 3(a)), and $[-4, -3.5]$ for an expansion on the left bound (dark red cone in Fig. 3(b)). For this problem, choosing a small slice leads to a large reduction (relatively to the size of the slice) for both expansion points. For some "well-behaved" functions, however, larger slices have the potential of larger reductions. The same dilemma arises in choosing the expansion point: Comparing Fig. 3(a) and Fig. 3(b), we see that for the slice $[-8, -3.5]$, an expansion point on the center offers a better reduction than on the left bound; The reverse is true for the slice $[-8, 0]$ (light red cones in Fig. 3(a) and Fig. 3(b)), where an expansion on the center leads to no reduction at all because the part to discard is strictly included in the domain considered. On the other hand, the expansion on the left bound always leads to some reduction, as promised above.

To shave the left part of a domain $[a, b]$, the algorithm presented by McAllester *et al.* [4] first considers the slice formed by the whole domain. If the reduction obtained by a Newton step does not lead to a reduction by 10% or more, it considers the left half $[a, (a+b)/2]$; again, if the reduction is insufficient, it considers the leftmost quarter of the initial domain, $[a, (3a+b)/4]$. Lastly, if the reduction is still insufficient, it considers the leftmost eighth, $[a, (7a+b)/8]$. At that point, it returns whatever domain $[l, b]$ was obtained, even if its left bound is not box consistent (i.e. $0 \notin \boldsymbol{g}([l, l^{+}])$). For all Newton steps performed, it uses the center of the domain under scrutiny as the expansion point, in a bid to achieve large reductions.

Following McAllester *et al.*'s work, we have designed an algorithm that uses a different shaving process to address its shortcomings (see discussion below). The corresponding left narrowing procedure $\mathit{lnar}_{\mathrm{sbc3ag}}$ is presented in Alg. 4

(The *rnar*$_{\text{sbc3ag}}$ procedure is not shown, since it can easily be obtained from *lnar*$_{\text{sbc3ag}}$ by symmetry). An important difference with McAllester *et al.*'s algorithm is that ours enforces box consistency every time, while theirs usually stops the narrowing process before that point. As Section 4 will show, spending more time to effectually enforce box consistency is the key to good performances for some difficult problems.

To shave the left part of a domain $[a, b]$, we proceed as follows (see Alg. 4): We consider a slice whose width is γ times the width W of the initial domain, and we apply one Newton step on it; if the reduction obtained is smaller than some threshold σ_b, we update γ to $\beta_b\gamma$, with $\beta_b < 1$; if the reduction is greater than some threshold σ_g, we update γ to $\beta_g\gamma$, with $\beta_g > 1$. Otherwise, we keep γ as it is. We then consider the next slice with a size equal to the new γ times W. In short, when we are able to shave large parts, we take larger bids; on the other hand, we decrease the size of the slices considered when things go wrong. This method follows the line of well-known numerical algorithms such as *Trust Region Methods*.

As shown in Alg. 4, the amount of reduction is computed by comparing the size of the previous domain with the size of the tightened domain where the right bound has been reset to the one of the previous domain (that is, we do not consider the possible displacement of the right bound as a useful reduction since it is not taken advantage of afterwards).

Another difference with McAllester *et al.*'s algorithm is that we use the left bound as an expansion point when shaving the left side (and the right bound when shaving the right side). There are two reasons for this: Firstly, this is a surefire strategy since it always lead to some reduction, as said previously; Secondly, we have to compute $\boldsymbol{g}([\underline{I}, \underline{I}^+])$ (Lines 3 and 20 in Alg. 4) before each Newton step to check whether the left bound is already box consistent. This value can be reused instead of $\boldsymbol{g}([\underline{I}, \underline{I}])$ in the computation of the Newton step Line 11 at a negligible cost in terms of the width of intervals involved (the correction of the process is obviously preserved since we use a wider interval), while avoiding the cost of one function evaluation. Unreported experimental evidences show that speed-ups up to 30% are attainable that way.

For the problem presented in Fig. 3, the original *bc3revise* algorithm [6] (that is, the one instantiated with *lnar*$_{\text{vhmak}}$ and *rnar*$_{\text{vhmak}}$) obtains the largest box consistent domain (approximately $[-0.487, 10]$) after 50 Newton steps; McAllester *et al.*'s algorithm stops with the non box consistent domain $[-3.5, 10]$ after only 6 Newton steps, and our algorithm enforces box consistency after 27 Newton steps.

4. EXPERIMENTS

To evaluate the impact of our algorithm, we have selected 16 instances of 14 classical test problems. Some are polynomial and others are not. The characteristics of these test problems are summarized in Table 1. All problems are structurally well constrained, with as many equations as variables. Column "Constraints" indicates whether all constraints are polynomial (quadratic, if no polynomial has a degree greater than 2). A problem is labelled "non-polynomial" if at least one constraint contains a trigonometric, hyperbolic or otherwise transcendental operator. Column "Source" gives a (not necessarily primary) reference to the literature where the problem is presented. The size of the problem and the initial domains for the variables are given in Table 2.

Algorithm 4 Enforcing box consistency on the left bound with adaptive guesses

```
[$\boldsymbol{lnar}_{\mathbf{sbc3ag}}$] in: $\boldsymbol{g}\colon \mathbb{I} \to \mathbb{I}$; in: $\boldsymbol{I} \in \mathbb{I}$
# Thresholds $\sigma_g$ and $\sigma_b$, and inflating/reducing factors
# $\beta_g$ and $\beta_b$ chosen experimentally.
const: $\gamma_{\text{init}} = 0.25, \sigma_g = 0.25, \sigma_b = 0.75, \beta_g = 1.5, \beta_b = 0.7$
begin
   1  $\gamma \leftarrow \gamma_{\text{init}}$
   2  $W \leftarrow \mathsf{w}(\boldsymbol{I})$ # $W$ fixed to avoid asymptotic behavior
   3  $\boldsymbol{d} \leftarrow \boldsymbol{g}([\underline{I}, \underline{I}^+])$
   4  # $\boldsymbol{I}$ is not box consistent on left?
   5  while $\boldsymbol{I} \neq \varnothing \wedge 0 \notin \boldsymbol{d}$:
   6     $\boldsymbol{I} \leftarrow [\underline{I}^+, \overline{I}]$ # We already know $0 \notin [\underline{I}, \underline{I}^+]$
   7     $\mathbf{guess} \leftarrow [\underline{I}, \min(\overline{I}, \underline{I} + \gamma W)]$
   8     if $0 \notin \boldsymbol{g}(\mathbf{guess})$
   9        $\mathbf{reduced} \leftarrow \varnothing$
  10     else: # Performing one Newton step
  11        $\mathbf{reduced} \leftarrow \mathbf{guess} \cap$
                 $\Big(\underline{\text{guess}} - \boldsymbol{d} \oslash \boldsymbol{g'}(\mathbf{guess})\Big)$
  12     if $\mathsf{w}([\underline{\text{reduced}}, \overline{\text{guess}}]) < \sigma_g \mathsf{w}(\text{guess})$:
  13        # Good reduction?...
  14        $\gamma \leftarrow \gamma\beta_g$ # ...Let us consider larger guesses
  15     elif $\mathsf{w}([\underline{\text{reduced}}, \overline{\text{guess}}]) > \sigma_b \mathsf{w}(\text{guess})$:
  16        # Bad reduction?...
  17        $\gamma \leftarrow \gamma\beta_b$ # ...Let us consider smaller guesses
  18     # We keep the right bound unchanged:
  19     $\boldsymbol{I} \leftarrow \Box\big(\mathbf{reduced} \cup (\boldsymbol{I} \setminus \mathbf{guess})\big)$
  20     $\boldsymbol{d} \leftarrow \boldsymbol{g}([\underline{I}, \underline{I}^+])$
  21  return $\boldsymbol{I}$
end
```

Table 1: Test problems

Name	Code	Constraints	Source
Broyden-banded	bb	quad.	[6]
Broyden tridiagonal	bt	quad.	[5]
Combustion	comb	poly.	[6]
DBVF	dbvf	poly.	[5]
Extended Freudenstein	ef	poly.	COPRIN[3]
Extended Powell	ep	poly	COPRIN[3]
Feigenbaum	fe	quad	[2]
i4	i4	poly.	[6]
Mixed Algebraic Trig.	mat	non-poly.	COPRIN[3]
Moré-Cosnard	mc	poly.	[6]
Trigexp	te	non-poly.	COPRIN[3]
Trigexp 3	te3	non-poly.	COPRIN[3]
Troesch	tro	non-poly.	COPRIN[3]
Yamamura	yam	poly.	COPRIN[3]

All experiments were conducted on an Intel Core2 Duo T5600 1.83GHz. The Whetstone test for this machine reports 1111 MIPS with a loop count equal to 100,000.

All algorithms have been implemented in our own C++ constraint solver, with the not-yet-released version 4.0 of *gaol*[4] as the underlying interval arithmetic library.

Table 2 reports the time spent in seconds to isolate all solutions of the test problems in domains with a width smaller than 10^{-8}, starting from the initial domains given in Column "Domains". The number preceded by a dot in Column "Problems" gives the number of equations and variables of the problem. An entry "TO" indicates a time-out (fixed to more than one hour, here). An entry "OV" indicates

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

[4]`http://sf.net/projects/gaol/`

Table 2: Experimental Results

Problems	Domains	Times (in seconds)				
		vhmak	mavhk$_m$	mavhk$_b$	sbc3ag$_b$	sbc3ag$_m$
bb.1000	$[-10^8, 10^8]$	**48.9**	**5.6**	19.3	**9.3**	43.2
bt.20	$[-100, 100]$	**121.6**	**25.8**	25.9	**21.1**	97.7
bt.30	$[-100, 100]$	***TO***	**2063.4**	1917.7	**1931.2**	*TO*
comb.10	$[-10^8, 10^8]$	**5.6**	**3.2**	*TO*	**5.2**	10.0
dbvf 200	$[-100, 100]$	**2343.1**	**655.1**	435.3	**553.8**	1697.2
ef.4000	$[-10^8, 10^8]$	**2031.5**	**1934.4**	*TO*	**920.2**	1101.1
ep.20	$[-10^8, 10^8]$	**4.9**	**465.1**	18.5	**3.0**	3.3
ep.30	$[-10^8, 10^8]$	**182.5**	***OV***	222.6	**78.7**	121.9
fe.20	$[-10^8, 10^8]$	**508.0**	**610.6**	233.9	**126.3**	477.2
i4.10	$[-1, 1]$	**4.2**	**4.6**	3.2	**2.0**	3.1
mat.3	$[-10^8, 10^8]$	**2.0**	**0.4**	0.5	**0.3**	1.6
mc.200	$[-10^8, 0]$	**297.3**	**246.5**	253.4	**214.8**	272.0
te.14	$[-100, 100]$	**196.5**	**84.5**	*TO*	**74.9**	202.7
te3.15000	$[0.36, 2.72]$	**6.1**	**3.6**	3.3	**3.0**	4.9
tro.400	$[-8, 8]$	**2676.9**	**718.0**	443.5	**484.3**	1931.8
yam.10	$[-10^8, 10^8]$	**538.5**	**161.3**	157.4	**150.4**	394.2

Times on an Intel Core2 Duo T5600 1.83GHz (whetstone 100 000: 1111 MIPS). Time-out set to 1 hour.

that an overflow (stack or heap) prevented the computation to complete. Column "vhmak" stands for Alg. 1 instantiated with $lnar_{\text{vhmak}}$ and $rnar_{\text{vhmak}}$ [6]; Column "mavhk$_m$" corresponds to McAllester *et al.*'s algorithm [4]; Column "mavhk$_b$" corresponds to McAllester *et al.*'s algorithm where the Newton expansion is done on the left and right bounds instead of the middle; Column "sbc3ag$_b$" corresponds to our adaptive algorithm (that is, *bc3revise*[$lnar_{sbc3ag}$,$rnar_{sbc3ag}$]); lastly, Column "sbc3ag$_m$" corresponds to our adaptive algorithm where the Newton expansion is done in the middle of the interval considered.

Except for "bb" and "comb," Alg. *sbc3ag$_b$* is faster than McAllester *et al.*'s algorithm, sometimes strikingly so (e.g., on extended-powell). Alg. *mavhk$_m$* outperforms it on Broyden Banded because this one is an easy problem with only one solution where it pays to be bold in the size of the slices considered for discarding, which is exactly what Alg. *mavhk$_m$* does. As evidenced by the number of splits on this benchmark (the same for all algorithms), the fact that Alg. *mavhk$_m$* may defer to the binary search process of the solver while stopping computation before box consistency is reached has no part in the performances of this algorithm here. On the other hand, Alg. *mavhk$_m$* achieves good performances on problems "com" and "te" while requiring, respectively, 5 times and 2.5 times more splittings than the other methods. This validates the argument by McAllester *et al.* that spending less time in each contracting operator may pay off.

Nevertheless, some problems (feigenbaum and extended-powell, among others) show that it sometimes pays off to try hard to tighten the domains at the constraint operator level rather than deferring to the dichotomic process: *sbc3ag$_b$* is much faster than *mavhk$_m$* on these benchmarks, and, tellingly, *mavhk$_m$* fares worse than *vhmak* as well.

Times-out for Alg. *mavhk$_b$* correspond to slow convergence phenomena as the one described in Fig. 2(b).

A comparison of Columns *sbc3ag$_b$* and *sbc3ag$_m$*, and of Columns *mavhk$_m$* and *mavhk$_b$* shows that the choice of point for a Newton expansion (middle point vs. bounds) does not explain alone the good performances of *sbc3ag$_b$*; even more, we see that using McAllester's et al. algorithm with an expansion on the bounds leads to an algorithm that may converge very slowly. With the extended-powell problem, we also see that an expansion on the bounds leads to regular performances, provided we either do not try to reach box consistency in order to avoid slow convergence (compare Alg. *mavhk$_b$* with *mavhk$_m$*), or we use adaptive guesses (Algs. *sbc3ag$_b$* and *sbc3ag$_m$*).

5. CONCLUSION

As Tab. 2 shows, and contrary to the statement by Benhamou *et al.* [1], using the bounds as expansion points for the Newton method is efficient, provided that the Newton operator is applied on a carefully chosen subpart of the initial domain and not on the whole of it as Benhamou *et al.* presumably tested.

Algorithm *sbc3ag$_b$* is significantly faster than McAllester *et al.*'s algorithm on a subset of the problems only. However, its strength lies elsewhere, viz. it is the most dependable algorithm among the ones tested here to enforce box consistency in that it is always only slightly slower than the best algorithm, while never exhibiting bad performances as McAllester *et al.*'s algorithm does on difficult problems.

6. REFERENCES

[1] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Procs. Intl. Symp. on Logic Prog.*, pages 124–138. The MIT Press, 1994.

[2] C. Jäger and D. Ratz. A combined method for enclosing all solutions of nonlinear systems of polynomial equations. *Reliable Computing*, 1(1):41–64, 1995.

[3] R. B. Kearfott, M. T. Nakao, A. Neumaier, S. M. Rump, S. P. Shary, and P. van Hentenryck. Standardized notation in interval analysis. In *Proc. XIII Baikal Int'l School-seminar "Optimization methods and their applications"*, volume 4 "Interval analysis", pages 106–113, 2005.

[4] D. A. McAllester, P. Van Hentenryck, and D. Kapur. Three cuts for accelerated interval propagation. Technical Memo AIM-1542, MIT, AI Lab., May 1995.

[5] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM TOMS*, 7(1):17–41, 1981.

[6] P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Num. Anal.*, 34(2):797–827, Apr. 1997.