CWI Quarterly Volume 11 (2&3) 1998, pp. 249 – 272

Chaotic Iteration for Polynomial Constraints

Eric Monfroy
CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
email: eric@cwi.nl

In this paper we argue for an alternative way of designing solvers based on interval arithmetic. We achieve constraint propagation over real numbers using chaotic iteration, a general and basic technique used for computing limits of iterations of finite sets of functions. This is carried out in two steps: first involving computationally "cheap" functions for reducing constraint satisfaction problems followed by computationally "expensive" functions for enforcing a local consistency property. This technique improves the global performance of the propagation mechanism in a simple way.

1. Introduction

During the last twenty years, many techniques were studied for solving Constraint Satisfaction Problems (CSP). *Constraint propagation* (one of the most important techniques for solving CSPs) consists in reducing a CSP into another one that is equivalent, but simpler. Since solving a CSP is in general computationally intractable, the algorithms for constraint propagation do not enforce global consistency, but local consistency (such as arc-consistency [15]) that "approximates" the global property.

For solving polynomial constraints over real numbers, several consistency methods have been designed [20]. In the scheme of interval constraints [13, 5, 3, 6, 21, 11, 14] propagation algorithms consist in enforcing a local consistency property during the whole reduction of CSPs (i.e., box-consistency [5] in [23, 22], B-consistency in [13]): domains of variables are reduced to obtain a local property (using a single constraint or a sub-set of constraints) and the modifications are propagated till reaching a fix-point. B-consistency requires preliminary transformations of CSPs that increase the number of constraints and variables and consequently lead to weaker reductions of CSPs. In an other hand, box-consistency (a restriction of arc-consistency to boundaries of intervals that is parametrized by an interval extension of the constraints) is costly as it requires a recursive local splitting of a domain at each reduction step.

Constraint propagation algorithms are instances of more general algorithms dealing with *chaotic iteration* [2], a basic technique used for computing limits of iterations of finite sets of functions. Local consistency (such as B or box-consistency) is generally described as a common fixed-point of domain reduction functions. Hence, by "feeding" these functions into a chaotic iteration algorithm, we generate an algorithm that achieves local consistency.

In this paper, we present constraint propagation over real numbers using chaotic iteration. We also propose to add weaker but cheaper domain reduction functions to preliminary reduce CSPs before enforcing box-consistency with expensive domain reduction functions: this mechanism does not modify the property but increases the global performances of the algorithm.

Due to space limitation, we restrict this paper to box-consistency with distributed interval extension and we present only four types of domain reduction functions (based on a single technique i.e., the interval Newton method [19]) that are from the weakest to the strongest: 1) a single application of Newton, 2) an idempotent version of Newton, 3) a function that limit the local splitting inherent to box-consistency, and 4) a function that achieves box-consistency [5].

Several strategies can be introduced in the chaotic iteration algorithm: the choice of the set of domain reduction functions, the selection of a function for reducing a domain, and the update of the current set of functions. A prototype validated the feasibility of our approach and enabled us comparing various combinations of strategies. Some experimental results showed that feeding the algorithm with weaker functions not only speeds up box-consistency computation, but also reduces the required amount of memory. Moreover, some extra reduction functions can be naturally added for treating particular constraints (such as linear or quadratic constraints, ...).

The paper is organized as follows. Section 2 introduces interval arithmetic, and interval extensions of constraints. Section 3 presents chaotic iteration for constraint propagation. In Section 4 we describe some domain reduction functions for solving constraints over real numbers. Section 5 presents the prototype for reducing CSPs over the real numbers, some comparisons of various strategies and some experimental results. Finally, comparisons with previous systems, conclusions and future works are discussed in Section 6.

2. Interval constraints

2.1. Intervals

Although some earlier works could be cited, interval mathematics really begun with R.E. Moore's book in 1966 [19].

We consider $\mathbb{R}^{\infty}$ the set of real numbers extended with the two infinity symbols $-\infty$ and ∞. We also consider the natural extension of the relation $<$

to $\mathbb{R}^\infty$. We consider $\mathcal{F}$ a subset of $\mathbb{R}$ containing $-\infty$, ∞ and 0 [1].

DEFINITION 2.1 (INTERVAL) *An interval $I = [a, b]$ consists of the set $\{x \in \mathbb{R} | a \leq x \leq b\}$ of real numbers where $a \in \mathcal{F}$ and $b \in \mathcal{F}$. The set of intervals I is denoted by $\mathcal{I}$ and is ordered by set inclusion.*

We denote by $\vec{I}$ the box of intervals $I_1 \times \cdots \times I_n$. I^L (respectively I^R) denotes the left (respectively right) endpoint of an interval. Thus, if $I = [a, b]$ then $a = I^L$ and $b = I^R$. Two intervals $I_1 = [a, b]$ and $I_2 = [c, d]$ are equal if and only if $a = c$ and $b = d$.

With respect to the definition, an interval is a set. Thus, we can associate some usual set operators to intervals. The intersection of two intervals is defined by:

$$I \cap J = \begin{cases} \emptyset & \text{if } I^L > J^R \text{ or } I^R < J^L \\ [max(I^L, J^L), min(I^R, J^R)] & \text{otherwise} \end{cases}$$

If the intersection of two intervals I and J is non-empty, their union is the interval:

$$I \cup J = [min(I^L, J^L), max(I^R, J^R)]$$

The union of two intervals that have an empty intersection is not an interval. In this case, we define the *interval hull* of two intervals I and J by:

$$I \uplus J = [min(I^L, J^L), max(I^R, J^R)]$$

These operations can be extended to boxes. Let $\vec{I} = I_1 \times \cdots \times I_n$ and $\vec{J} = J_1 \times \cdots \times J_n$ be two boxes. Then:

$$\vec{I} \cap \vec{J} = (I_1 \cap J_1) \times \ldots \times (I_n \cap J_n)$$

and

$$\vec{I} \uplus \vec{J} = (I_1 \uplus J_1) \times \ldots \times (I_n \uplus J_n)$$

Let x be a number of $\mathcal{F}$. We denote by x^+ the smallest number of $\mathcal{F}$ greater than x, and by x^-, the largest number of $\mathcal{F}$ smaller than x.

Since an interval $[a, b]$ can be the result of a computation on a computer, we have to use *outward rounding*. This process consists in rounding a (respectively b) to the largest (respectively smallest) element of $\mathcal{F}$ which is less (respectively greater) than or equal to a (respectively b). In order to perform outward rounding, we have to do *directed rounding*. This is defined in the IEEE standard for floating point arithmetic [12], such an algorithm can also be found in [1]. In the following, $\lfloor x \rfloor$ represents the greatest element of $\mathcal{F}$ smaller than or equal to

[1] In practice, $\mathcal{F}$ denotes the ordered set of floating point numbers; ∞ (respectively $-\infty$) represents the largest (respectively the smallest) floating point number with respect to the implementation.

the real number x, and $\lceil x \rceil$ represents the smallest element of $\mathcal{F}$ greater than or equal to x. We denote by $\overline{x}$ the interval $[\lfloor x \rfloor, \lceil x \rceil]$.

Let I be a given interval. We define its *center* or *midpoint* by $m(I) = \lfloor (I^L + I^R)/2 \rfloor$. The *width* of I is $w(I) = \lceil I^R - I^L \rceil$. The width of an interval will be the measure to compare different computations with the same input. When we say that *the precision is increased* (or *the accuracy is better*), this means that the width of the result is smaller than the result computed with another method.

2.2. Interval arithmetic

Assume that $\mathcal{F} = \mathbb{R}^\infty$. Let $+, -, *, /$, and $\hat{}$ be the usual operations of addition, subtraction, multiplication, division and exponentiation over the real numbers. Let $op \in \{+, -, *, /, \hat{}\}$ denote one of these operations. Then, op_{int}, the extension of op to the arithmetic of interval numbers, is defined by:

$$I \ op_{int} \ J = \{x \ op \ y : x \in I, y \in J\}$$

where I and J are intervals of real numbers.

If we consider outward rounding, we obtain extensions that have the following weaker property:

$$I \ op_{int} \ J \supseteq \{x \ op \ y : x \in I, y \in J\}$$

Let assume outward rounding. We now give the rules (as there are given in [8]) for computing the endpoints of $I \ op_{int} \ J$ with $I = [a, b]$ and $J = [c, d]$. Since no confusion is possible, we denote operators over $\mathcal{F}$, and their extensions by the same symbol.

- **addition:**

$$I + J = [\lfloor a + c \rfloor, \lceil b + d \rceil]$$

- **subtraction:**

$$I - J = [\lfloor a - d \rfloor, \lceil b - c \rceil]$$

- **multiplication:**

$$I * J = [\lfloor min(a * c, a * d, b * c, b * d) \rfloor, \lceil max(a * c, a * d, b * c, b * d) \rceil]$$

The extension of $*$ can be refined using a case analysis:

$$I * J = \begin{cases} [\lfloor a*c \rfloor, \lceil b*d \rceil] & \text{if } a \geq 0 \text{ and } c \geq 0 \\ [\lfloor b*c \rfloor, \lceil b*d \rceil] & \text{if } a \geq 0 \text{ and } c < 0 < d \\ [\lfloor b*c \rfloor, \lceil a*d \rceil] & \text{if } a \geq 0 \text{ and } d \leq 0 \\ [\lfloor a*d \rfloor, \lceil b*c \rceil] & \text{if } a < 0 < b \text{ and } c \geq 0 \\ [\lfloor b*d \rfloor, \lceil a*d \rceil] & \text{if } a < 0 < b \text{ and } d \leq 0 \\ [\lfloor a*d \rfloor, \lceil b*c \rceil] & \text{if } b \leq 0 \text{ and } c \geq 0 \\ [\lfloor a*d \rfloor, \lceil a*c \rceil] & \text{if } b \leq 0 \text{ and } c < 0 < d \\ [\lfloor b*d \rfloor, \lceil a*c \rceil] & \text{if } b \leq 0 \text{ and } d \leq 0 \\ [\lfloor min(b*c, a*d) \rfloor, & \text{if } a < 0 < b \text{ and } c < 0 < d \\ \quad \lceil max(a*c, b*d) \rceil] & \end{cases}$$

– **division:**

$1/J = [\lfloor 1/d \rfloor, \lceil 1/c \rceil]$ if $0 \notin$ J

and

$I/J = I * (1/J)$ if $0 \notin$ J

However, these rules can be refined with *extended interval arithmetic* in order to take into account division by 0:

$$I/J = \begin{cases} [\lfloor b/c \rfloor, \infty] & \text{if } b \leq 0 \text{ and } d = 0 & (1) \\ [-\infty, \lceil b/d \rceil] \cup [\lfloor b/c \rfloor, \infty] & \text{if } b \leq 0 \text{ and } c < 0 < d & (2) \\ [-\infty, \lceil b/d \rceil] & \text{if } b \leq 0 \text{ and } c = 0 & (3) \\ [-\infty, \infty] & \text{if } a < 0 < b & (4) \\ [-\infty, \lceil a/c \rceil] & \text{if } a \geq 0 \text{ and } d = 0 & (5) \\ [-\infty, \lceil a/c \rceil] \cup [\lfloor a/d \rfloor, \infty] & \text{if } a \geq 0 \text{ and } c < 0 < d & (6) \\ [\lfloor a/d \rfloor, \infty] & \text{if } a \geq 0 \text{ and } c = 0 & (7) \end{cases}$$

Since we do not want to use rules of extended interval arithmetic that yield sequences of intervals, rules (2) and (6) are replaced by rules (2′) and (6′) as defined below. So the union of intervals ($\cup$) is replaced here by the interval hull ($\uplus$).

$$\begin{array}{lll} [-\infty, \infty] & \text{if } b \leq 0 \text{ and } c < 0 < d & (2') \\ [-\infty, \infty] & \text{if } a \geq 0 \text{ and } c < 0 < d & (6') \end{array}$$

However, in Section 4.1, we have to evaluate numerous functions of the form $I - J/K$ and to intersect the result with an interval L. At this stage, and only for this special purpose, we use (when the required conditions are fullfilled) the rules (2) and (6) of extended interval arithmetic to compute J/K. J/K is then subtracted from I using extended interval arithmetic. Finally, when J/K returns a union of intervals, the intersection of $I - J/K$ with L is the interval hull of all the intersections of the result of $I - J/K$ with L. This process aims at increasing the precision of the result: locally, we consider union of intervals, but globally, we obtain only one interval.

– **Infinite or semi-infinite intervals:**

$$\begin{array}{l}[a,b]+[-\infty,d]=[-\infty,b+d]\\ [a,b]+[c,\infty]=[a+c,\infty]\\ [a,b]\pm[-\infty,\infty]=[-\infty,\infty]\\ [a,b]-[-\infty,d]=[a-d,\infty]\\ [a,b]-[c,\infty]=[-\infty,b-c]\end{array}$$

– **power:** Although I^n could be defined by $\underbrace{I * \ldots * I}_{n}$, the following rules increase precision (see the dependency problem in Section 2.4):

$$I^n = \begin{cases} [1,1] & \text{if } n=0 \\ [\lfloor a^n \rfloor, \lceil b^n \rceil] & \text{if } a \geq 0 \text{ or } a \leq 0 \leq b \text{ and } n \text{ is odd} \\ [\lfloor b^n \rfloor, \lceil a^n \rceil] & \text{if } b \geq 0 \\ [0, \lceil max(a^n, b^n) \rceil] & \text{if } a \geq 0 \geq b \text{ and } n \text{ is even} \end{cases}$$

2.3. Interval functions and constraints

We now extend interval arithmetic to arbitrary functions. For this purpose, we need the notion of *interval extension*.

DEFINITION 2.2 (INTERVAL EXTENSION OF FUNCTION) *Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a real-valued function. $F : \mathcal{I}^n \rightarrow \mathcal{I}$ is an interval extension of f if and only if:*

$$\forall I_1 \cdots I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow f(r_1, \cdots, r_n) \in F(I_1, \cdots, I_n).$$

DEFINITION 2.3 (INCLUSION MONOTONIC) *An interval function F is said to be inclusion monotonic if $X_1 \subset Y_1, \ldots, X_n \subset Y_n$ implies $F(X_1, \cdots, X_n) \subset F(Y_1, \cdots, Y_n)$.*

It follows from the definitions of op and op_{int} that interval arithmetic is inclusion monotonic [1], that is, if op_{int} represents $+, -, *, /$ or $\hat{}$, then $X_1 \subset Y_1$, and $X_2 \subset Y_2$ implies $(X_1 \; op_{int} \; X_2) \subset (Y_1 \; op_{int} \; Y_2)$.

We now define what is the extension of a real constraint, i.e. a relation from $\mathbb{R}^n$ to the set of booleans.

DEFINITION 2.4 (INTERVAL EXTENSION OF CONSTRAINT) *Let $c : \mathbb{R}^n \rightarrow \mathcal{B}$ be a real constraint. $C : \mathcal{I}^n \rightarrow \mathcal{B}$ is an interval extension of c if and only if:*

$$\forall I_1 \cdots I_n \in \mathcal{I} \; [(\exists r_1 \in I_1, \ldots, \exists r_n \in I_n \; c(r_1, \cdots, r_n)) \Rightarrow C(I_1, \cdots, I_n)].$$

The interval extension of the equality constraints over reals [2], $=$ can be defined by:

[2] To simplify notation, we often use the same symbol for an arithmetic operation or a constraint and its interval extension.

$I_1 = I_2 \;\Leftrightarrow\; I_1 \cap I_2 \neq \emptyset$

The arithmetic extensions of the previous section are all interval extensions. It is important to notice that interval extension of a function is not unique. Thus, different extensions lead to different accuracy of the result (i.e., the width of the result of an extension is smaller than the width of the result of another one). However, in general, the more accurate is the extension, the more complex is the computation. This motivates the next two sections where the problem of dependencies between variables, and several interval extensions are described.

2.4. Accuracy and the dependency problem

For simplicity reasons, we assume an undefined interval arithmetic (without rounding). Using the previous rules, the evaluation of $I-I$ gives $[I^L - I^R, I^R - I^L]$ while one would expect $\overline{0}$. In fact, we have computed $\{x-y : x \in I, y \in I\}$ ($\supseteq I-I$) and not $\{x-x : x \in I\}$. Informally, we "lost" the dependency between the two occurences of I. This problem is known in interval arithmetic as the *dependency problem.*

Assume now three equivalent functions $f_1 = x^2 - x$, $f_2 = x(x-1)$, and $f_3 = (x-1/2)^2 - 1/4$, and some interval extensions $F_1 = X^2 - X$, $F_2 = X(X-\overline{1})$, and $F_3 = (X-\overline{1/2})^2 - \overline{1/4}$, of f_1, f_2, and f_3 respectively. Consider the following evaluations with $X = [0,2]$:

$$F_1([0,2]) = [0,2]^2 - [0,2] = [0,4] - [0,2] = [-2,4]$$

$$F_2([0,2]) = [0,2] * ([0,2] - [1,1]) = [0,2] * [-1,1] = [-2,2]$$

$$\begin{aligned} F_3([0,2]) &= ([0,2] - [1/2,1/2])^2 - [1/4,1/4] = [-1/2,3/2]^2 - [1/4,1/4] \\ &= [0,9/4] - [1/4,1/4] = [-1/4,2] \end{aligned}$$

We obtain $F_3([0,2]) \subseteq F_2([0,2]) \subseteq F_1([0,2])$. Thus, the syntaxic form of the functions is also important for the accuracy of their extensions [19]. Notice that in F_3, there is no more dependency problem as X occurs only once, and the result is exact.

2.5. Some interval extensions

In the following, we consider fixed interval arithmetic (as defined in [1, 8], and in Section 2.2). As we have just noticed, the syntaxic form of the function significantly influences the interval obtained after its evaluation. However, functions can be processed before evaluation in order to get more accurate results.

Natural interval extension This extension is the most simple one. It consists in replacing each element of the function by its direct extension (i.e. a number a is replaced by $\overline{a}$, a real variable x is replaced by an interval variable X, real arithmetic operations by their corresponding interval extensions, and constraints by their extensions).

For example, let $x_1 * (x_2 + 1.3 + x_1) = 4$ be a real constraint. Its natural interval extension is the interval constraint $X_1 * (X_2 + \overline{1.3} + X_1) = \overline{4}$.

With the natural interval extension, constraints maintain the syntaxic form. Thus, to improve accuracy one can minimize dependencies. This extension is generally more accurate than the next one. Moreover, we can also consider factorizing the polynomials (however, this operation is expensive).

Distributed interval extension The *distributed interval extension* consists in applying the *natural interval extension* to the distributed form (i.e., syntactical form) of a function. Distributing a function consists in giving a particuliar syntactic form of the function.

Let f be a function, then we say that f is in a *distributed form* if f is written as

$$\sum_{i=1..n} \left(c_i \prod_{j_i=1..m} x_{j_i}^{e_{j_i}} \right)$$

where $\forall i\ c_i \in \mathbb{R}$ and $\forall i, j\ e_{j_i} \in \mathbb{N}$.

The distributed form of a constraint $c(x_1, \cdots, x_n)$ is straight forward: replacing $x_1, \cdots, x_n$ by their distributed form is sufficient to obtain the distributed form of c. Then, the distributed interval extension of c consists in applying the natural interval extension to the distributed form of c.

Let $x_1 * (x_2 + 1.3 + x_1) = 4$ be a real constraint. Its distributed form is $x_1 * x_2 + 1.3 * x_1 + x_1^2 = 4$, and its distributed natural extension is $X_1 * X_2 + \overline{1.3} * X_1 + X_1^2 = \overline{4}$.

Due to lost of dependencies (see Section 2.4 for examples), the distributed interval is less accurate than some other extensions (in particular the natural interval extension) [19]. However, the distributed form enables us to perform an easier form of pre-evaluation (see Section 5.1). Thus, computations are more efficient when using several times a function, and more especially when some of the parameters are fixed as inside a reduction phase (see Section 4).

Taylor interval extension Taylor interval extension is a particuliar case of centered form [19]. This extension requires that the constraint is an equality of the form $f(x_1, \cdots, x_n) = 0$. It is also assumed that f has continuous derivatives of any necessary order with respect to each variable x_i. Informally, a Taylor expansion of f is applied around the center of the box $I_1, \cdots, I_n$. The rest of the series is bound using $I_1, \cdots, I_n$.

More precisely, let c be a constraint $f(x_1, \cdots, x_n) = 0$ such that f is a function with continuous partial derivatives in $x_1, \cdots, x_n$. The Taylor interval extension of c over the box $(I_1, \cdots, I_n)$ is the interval constraint:

$$F(\overline{m(I_1)}, \ldots, \overline{m(I_n)}) + \sum_{i=1..n} \frac{\delta F}{\delta X_i}(I_1, \cdots, I_n)(X_i - \overline{m(I_i)}) = \overline{0}$$

where $m(I)$ denotes the center of the interval I. This extension gives sharper interval bounds than the natural and distributed extension. However, it is also more complex and it leads to additional computing. Thus, the solution consists in using Taylor interval extension when more accurate intervals are really required, i.e., when endpoints of intervals are close to solutions of the constraint system.

Note that arithmetic interval extension as defined in [1, 8] and in Section 2.2 is inclusion monotonic, and consequently, natural, distributed and Taylor interval extensions are also inclusion monotonic.

3. Chaotic iteration and CSP

We now consider *constraint propagation*, one of the most important techniques used for solving Constraint Satisfaction Problems (CSP).Informally, constraint propagation consists in reducing a CSP into another one that is equivalent, but simpler.

3.1. Chaotic iteration

Constraint propagation algorithms are instances of more general algorithms dealing with *chaotic iteration* [2], a basic technique used for computing limits of iterations of finite sets of functions.

Definition 3.1 (Chaotic iteration) *Consider a set D, an element $d \in D$, and a set of functions $F = \{f_1, \cdots, f_k\}$ on D.*

- *a* run *of the functions $f_1, \cdots, f_k$ is an infinite sequence of numbers from $[1, k]$,*
- *a run $i_1, i_2, \ldots$ is* fair *if every $i \in [1, k]$ appears in it infinitely often,*
- *an* iteration *of F associated with a run $i_1, i_2, \ldots$ and starting with d is an infinite sequence of values $d_0, d_1, d_2, \ldots$ defined as follows: $d_0 = d$, and $d_j = f_{i_j}(d_{j-1})$.*
- *an iteration of F is* chaotic *if it is associated with a fair run.*

3.2. CSP

A domain membership is a relation of the form $x_1 \in D_{x_1}$. As no ambiguity is possible, we reduce this notation to D_1.

A *scheme* on n is a sequence of different elements from $[1, n]$. We say that C is a constraint on $\mathcal{D} = D_1, \cdots, D_n$ with scheme $i_1, \cdots, i_l$ if $C \subseteq D_{i1} \times \cdots \times D_{il}$.

A CSP $\langle \mathcal{D}; \mathcal{C} \rangle$ is defined by a sequence of domains $\mathcal{D}$ together with a sequence of constraints $\mathcal{C}$ on $\mathcal{D}$.

Given a n-tuple $d = d_1, \cdots, d_n$, and a scheme $s = i_1, \cdots, i_l$ on n, we denote by $d[s]$ the l-tuple $d_{i1}, \cdots, d_{il}$. A *solution* to a CSP $\langle D_1, \cdots, D_n; \mathcal{C} \rangle$ is a n-tuple $d \in D_1 \times \cdots \times D_n$ such that for each C in $\mathcal{C}$ with scheme s, $d[s] \in C$. We say that two CSPs are equivalent if they have the same set of solutions.

3.3. Domain reduction

We now focus on the aspects of chaotic iteration that are relevant for constraint propagation and domain reduction. We use the results developed by K. Apt in [2].

Consider a CSP $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$.

DO, the *domain associated with* $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ is the set

$$\{X_1 \times \cdots \times X_n \mid X_i \subseteq D_i \text{ for } i \in [1, n]\}$$

Let $s = i_1, \cdots, i_l$ be a scheme. We denote by DO_s the set

$$\{X_1 \times \cdots \times X_l \mid X_i \subseteq D_{j_i} \text{ for } i \in [1, l]\}$$

The set inclusion and intersection are extended to elements of DO_s. Let $X_1 \times \cdots \times X_l$ and $Y_1 \times \cdots \times Y_l$ be two elements of DO_s. Then:

- $X_1 \times \cdots \times X_l \subseteq Y_1 \times \cdots \times Y_l$ iff $X_i \subseteq Y_i$ for $i \in [1, l]$
- $(X_1 \times \cdots \times X_l) \cap (Y_1 \times \cdots \times Y_l) = (X_1 \cap Y_1) \times \ldots \times (X_l \cap Y_l)$

Let $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ and $\langle D'_1, \cdots, D'_n; \mathcal{C}\rangle$ be two CSPs. $\langle D'_1, \cdots, D'_n; \mathcal{C}\rangle$ is smaller than $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ if $D'_i \subseteq D_i$ for all $i \in [1, n]$.

We now consider functions on CSPs: they apply on a "part" of the domain $\mathcal{D}$ to modify some domain memberships of $\mathcal{D}$.

Definition 3.2 (Domain reduction function) *Consider a sequence of domains $D_1, \cdots, D_n$, and a constraint C with scheme s on n. A domain reduction function for C is a function on DO_s such that for all $D \in DO_s$*

- $f(D) \subseteq D$,
- $C \cap D = C \cap f(D)$.

The first condition states that no solution to C is gained (domains associated with C are reduced), and the second condition ensures that no solution to C is lost.

Let $DO = D_1, \cdots, D_n$ be a sequence of domains and $s = i_1, \cdots, i_l$ be a scheme on n. Consider a domain reduction function f on DO_s and suppose that $f(D_{i1} \times \cdots \times D_{il}) = D'_{i1} \times \cdots \times D'_{il}$. f on DO_s is extended to a function f^+ on D as follows:

$$f^+(D_1 \times \cdots \times D_n) = D''_1 \times \cdots \times D''_n$$

such that for all $i \in [1, n]$, $D''_i = D'_i$ if i is an element of s, $D''_i = D_i$ otherwise.

Consider a CSP $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$, a scheme s on n, and a domain reduction function f for the constraint C of $\mathcal{C}$ on DO_s. Suppose $f^+(D_1, \cdots, D_n) = D'_1, \cdots, D'_n$. Then, the CSP defined by $\langle D'_1, \cdots, D'_n; \mathcal{C}\rangle$ is a transformation of $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$, is smaller than $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$, and is equivalent to $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$.

Before stating the domain reduction theorem, a definition related to domain reduction function is needed.

DEFINITION 3.3 (MONOTONICITY, IDEMPOTENCY) *Consider a sequence of domains D, and a domain reduction function f on C with scheme s. The function f is called:*

– *monotonic if: $D \subseteq D'$ implies $f(D) \subseteq f(D')$ for all $D, D' \in DO_s$,*

– *idempotent if: $f(f(D)) = f(D)$ for all $D \in DO_s$,*

THEOREM 3.1 (DOMAIN REDUCTION) *Consider a CSP $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ and a set $F = \{f_1, \cdots, f_k\}$, where each f_i is a monotonic domain reduction function for some C in $\mathcal{C}$. Then,*

– *the limit of every chaotic iteration of $F^+ = \{f_1^+, \cdots, f_k^+\}$ exists and is equal to*

$$\bigcap_{j=0}^{\infty} f^j(D_1 \times \cdots \times D_n),$$

where f on DO is defined by:

$$f(D) = \bigcap_{i=1}^{k} f_i^+(D),$$

– *the CSP determined by $\langle \bigcap_{j=0}^{\infty} f^j(D_1 \times \cdots \times D_n); \mathcal{C}\rangle$ is equivalent to the CSP $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$.*

Informally, this theorem states that the order for applying the domain reduction functions has no importance, as long as fairness is kept. Furthermore, the computed limit is equivalent to the original CSP.

PROPERTY 3.4 (TERMINATION) *Suppose that all the domains D_i are finite. Then, the limit of every chaotic iteration of F^+ equals the largest fixed point of f with respect to set inclusion.*

This property (which is essential for interval arithmetic) also implies that the CI algorithm (see Figure 1) can be used to compute the limit of chaotic iteration of the domain reduction theorem. The input of the *generic chaotic iteration algorithm* is a CSP $\langle \mathcal{D}; \mathcal{C}\rangle$ together with a set F of functions on CSPs. The output is a CSP $\langle \mathcal{D}'; \mathcal{C}\rangle$ equivalent to $\langle \mathcal{D}; \mathcal{C}\rangle$ and smaller than $\langle \mathcal{D}; \mathcal{C}\rangle$.

4. DOMAIN REDUCTION FOR INTERVAL CONSTRAINTS

We now return to the problem of solving CSPs of the type $\langle \mathcal{D}; \mathcal{C}\rangle$, where:

– all the constraints C of $\mathcal{C}$ are equalities of polynomials,

– the domains D_i are intervals of $\mathcal{I}$, and each D_i is associated with a variable X_i,

```
CI
input $\langle D; C\rangle : CSP$; $F$: set of reduction functions
output $\langle D'; C\rangle : CSP$

$G := F$
while $G \neq \emptyset$
do
    select $g$ in $G$; suppose $g$ is with scheme $s$
    $G := G - \{g\}$
    $D'[s] := g(D[s])$
    if $D[s] \neq D'[s]$ then
      $G := G \cup \{f \in F | f$ depends on some $i$ in $s$ s.t. $D[i] \neq D'[i]\}$
      $D[s] := D'[s]$
    endif
enddo
```

FIGURE 1. Chaotic iteration algorithm

– $\mathcal{I}$ is the ordered set (with respect to set inclusion) of floating point intervals i.e. intervals of real numbers the bounds of which are elements of the finite set $\mathcal{F}$, the ordered set of floating-point numbers.

In the following, we always consider that we are working with a specific CSP $\langle \mathcal{D}; \mathcal{C}\rangle$, and that each variable is associated with a domain. We are now looking for domain reduction functions, i.e., functions that respects the following properties:

1: the functions are domain reduction functions for some $C \in \mathcal{C}$ with scheme s i.e., $f(\vec{I}) \subseteq \vec{I}$ and $C \cap \vec{I} = C \cap f(\vec{I})$ for all $\vec{I} \in DO_s$,

2: the functions are monotonic i.e. $\vec{I} \subseteq \vec{I'}$ implies $f(\vec{I}) \subseteq f(\vec{I'})$ for all $\vec{I}, \vec{I'} \in DO_s$.

REMARK 4.1. *Applications of most of the domain reduction functions that are presented in the next sections do not depend on the initial domains defined by the CSP. Thus we could replace DO_s by $\mathcal{I}^k$ (where k is the number of elements of the scheme s) in the previous requirements. However, some functions can be applied only when the constraint has some required properties on the initial domain (for example monotonically increasing). In order to get only one set of requirements for reduction functions, we keep DO_s.*

Since some reduction functions only apply to univariate constraints, we use projections to transform a multivariate constraint into a univariate one.

DEFINITION 4.2 (PROJECTION) *Let $\langle \mathcal{D}; \mathcal{C} \rangle$ be a CSP with $\mathcal{D} = D_1, \cdots, D_n$, C be a constraint of $\mathcal{C}$. Then C_{X_i}, the projection of C over X_i, is obtained from C by substituting the variables X_j by the intervals D_j, for all $j \neq i$.*

4.1. The interval Newton method

The interval Newton method [19] is an extension of the Newton method for finding roots of univariate functions. Although it was originally defined for univariate functions, we can also use the interval Newton method for multivariate functions.

Consider a univariate function f, its natural extension F, and an interval $I \in \mathcal{I}$. The interval Newton method applies a function $Newton(f, I)$ that reduces I to I' in such a way that all the zeros of f in I are also in I'. The function $Newton(f, I)$ is defined as follows:

$$Newton(f, I) = I \cap (\overline{a} - \frac{F(\overline{a})}{F'(I)})$$

where

- a is a point of I [3],
- F is an interval extension of f,
- F' an interval extension of the derivative of f.

We now consider the case of multivariate functions. Consider a function f on $x_1, \cdots, x_k$, its natural extension F, a domain $D_j \in \mathcal{I}$ for each variable x_j, $j \in [1, i-1] \cup [i+1, n]$, and an interval $I \in \mathcal{I}$. The interval Newton method applies a function $Newton(x_i, f, I)$ to reduce I into I', such that all the zeros of f in $D_1 \times \cdots \times D_{i-1} \times I \times D_{i+1} \times \cdots \times D_n$ are also in $D_1 \times \cdots \times D_{i-1} \times I' \times D_{i+1} \times \cdots \times D_n$:

$$Newton(x_i, f, I) = I \cap (\overline{a} - F_{X_i}(\overline{a}) / F'_{X_i}(I))$$

where

- a is a point of I,
- F_{X_i} is the projection on X_i of an interval extension of f,
- F'_{X_i} is the projection on X_i of an interval extension of the partial derivative of f with respect to x_i.

PROPERTY 4.3. *Let $\langle D_1, \cdots, D_n; \mathcal{C} \rangle$ be a CSP, $f(\vec{x}) = 0$ be a constraint of $\mathcal{C}$ with scheme s, and i an element of s. The function N_i defined as follows:*

$$N_i(D_1 \times \cdots \times D_n) = D_1 \times \cdots \times D_{i-1} \times D'_i \times D_{i+1} \times \cdots \times D_n$$

[3] a can be any point of I. However, it is convenient to choose the center of I [8].

where $D'_i = Newton(x_i, f, D_i)$, is a monotonic domain reduction function for the constraint $f(\vec{x}) = 0$ with scheme s on DO_s.
$\mathcal{N} = \{N_i$ for $C |$ for all C in $\mathcal{C}$ with scheme s, and for all i in $s\}$ is a set of monotonic domain reduction functions.

Reduction functions of $\mathcal{N}$ are not idempotent. We now consider iterative applications of $Newton(x_i, f, I)$, denoted $Newton^\star(x_i, f, I)$. Let $I_0 = I$ be the initial interval, we define step $n+1$ with respect to step n: $I_{n+1} = Newton(x_i, f, I_n)$. Hence, $Newton^\star(x_i, f, I) = I'$ where I' is a fixed-point of $Newton$.

Property 4.4. *$N_i^\star$ defined by*[4]

$$N_i^\star(D_1 \times \cdots \times D_n) = D_1 \times \cdots \times D_{i-1} \times D'_i \times D_{i+1} \times \cdots \times D_n$$

where $D'_i = Newton^\star(x_i, f, D_i)$, is a monotonic idempotent domain reduction function for the constraint $f(\vec{x}) = 0$ with scheme s on DO_s.
$\mathcal{N}^\star = \{N_i^\star$ for $C |$ for all C in $\mathcal{C}$ with scheme s, and for all i in $s\}$ is a set of monotonic domain reduction functions.

Remark 4.5. *Rules (2) and (6) from extended interval arithmetic for division are useful for functions of $\mathcal{N}$ and $\mathcal{N}^\star$, since the expressions to be computed in the Newton function are of the form $I' \cap (I - J/K)$.*

Some more complex reduction functions that reduce all the variables of a constraint in one step can be defined. However, they allows us for less control in chaotic iteration.

4.2. Box-consistency

The previously defined reduction functions enable to reduce a CSP $\langle \mathcal{D}; \mathcal{C} \rangle$ into a smaller CSP $\langle \mathcal{D}'; \mathcal{C} \rangle$. Although the set of solutions is preserved, no property is ensured: (1) the existence of solution is not guaranteed, and (2) if they are some solutions, we do not know where they are in the box $\mathcal{D}'$.

Arc-consistency [15] (a notion that gives some global properties to CSPs) is not well suited for non-linear constraints [5]: it would require splitting up the original CSP into too many sub-CSPs, one for each solution. Thus, we use the weaker notion of *box-consistency* [5].

Definition 4.6 (Box-consistency) *Let $\langle D_1, \cdots, D_n; \mathcal{C} \rangle$ be a CSP, and C a constraint from $\mathcal{C}$ with scheme s. Then, C is box-consistent with respect to i of s and $D_1, \cdots, D_n$ iff:*

$$C_{X_i}([D_i^R, D_i^{R^+}]) \wedge C_{X_i}([D_i^{L^-}, D_i^L])$$

[4] $N_i^\star$ can also be defined by $N_i^\star(DO_s) = \bigcap_{j=1}^{\infty} N_i^j(DO_s)$

where $D_i^{R^+}$ is the smallest number of $\mathcal{F}$ greater than D_i^R and $D_i^{L^-}$ is the largest number of $\mathcal{F}$ smaller than D_i^L.

C is box-consistent if it is box-consistent with respect to each i in s. A CSP $\langle \mathcal{C}; \mathcal{D}\rangle$ is box-consistent if each C in $\mathcal{C}$ is box-consistent[5].

The idea consists in using Newton to compute the leftmost zero and the rightmost zero of projections of constraints. For this purpose, we use a splitting mechanism which is local to the function and does not affect the CSP.

```
**i_Box_Consistency**
**input** $f : \mathbb{R}^n \to \mathbb{R};\ D_1 \times \cdots \times D_n : \mathcal{I}^n;\ i : \mathbb{N}$
**output** $D_i'' : \mathcal{I}$

    $D_i'$=pushleft$(f, D_1 \times \cdots \times D_n, i)$
    $D_i''$=pushright$(f, D_1 \times \cdots \times D_{i-1} \times D_i' \times D_{i+1} \times \cdots \times D_n, i)$
```

FIGURE 2. i_Box_Consistency

The function i_Box_Consistency (Figure 2) takes as input a function f on $x_1, \cdots, x_n$, a box of intervals $D_1 \times \cdots \times D_n$, and an index i (associated to a variable), and returns a new interval D_i' such that:

- $D_i' \subseteq D_i$,
- and $f(\vec{x}) = 0$ is box consistent with respect to i and the sequence of domains $D_1, \cdots, D_{i-1}, D_i', D_{i+1}, \cdots, D_n$.

The function pushleft (Figure 3) takes as input a function f, a box $D_1 \times \cdots \times D_n$, and an index i, and returns a new interval D_i' s.t. $D_i' \subseteq D_i$ and the leftmost solution (with respect to i-th coordinate) of $f(\vec{x}) = 0$ in $D_1 \times \cdots \times D_n$ is also in $D_1 \times \cdots \times D_{i-1} \times [D_i'^L, D_i'^{L^+}] \times D_{i+1} \times \cdots \times D_n$.

The split function splits an interval $[a, b]$ into two intervals $[a, c]$ and $[c, b]$[6].

Pushright is analogous to Pushleft and it deals with rightmost solution of $f(\vec{x}) = 0$.

PROPERTY 4.7. *Let $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ be a CSP, $f(\vec{x}) = 0$ a constraint of $\mathcal{C}$ with scheme s, and i an element of s. The function IBC_i defined as follows*

$$IBC_i(D_1 \times \cdots \times D_n) = D_1 \times \cdots \times D_{i-1} \times D_i' \times D_{i+1} \times \cdots \times D_n$$

where $D_i' = $ i_Box_Consistency$(f, D_1 \times \cdots \times D_n, i)$, is a monotonic domain reduction function for the constraint $f(\vec{x}) = 0$.
We put $\mathcal{IBC} = \{IBC_i$ for $C|$ for all C of the form $f(\vec{x}) = 0$ in $\mathcal{C}$ with scheme s, and for all i in $s\}$ is a set of monotonic domain reduction functions.

[5] Box-consistency depends on the interval extension (natural, distributed, Taylor, ...) that is used.
[6] We take c as being the center of $[a, b]$.

```
pushleft
input $f : \mathbb{R}^n \to \mathbb{R};\ D_1 \times \cdots \times D_n : \mathcal{I}^n;\ i : \mathbb{N}$
output $D'_i : \mathcal{I}$

$I := D_i$
if $0 \notin F_{X_i}(I)$
|  then $D'_i = \emptyset$
|  else $I' = Newton^\star(x_i, f, I)$
|        if $I' = \emptyset$
|        |  then $D'_i = \emptyset$
|        |  else if $0 \in F_{X_i}([I'^L, {I'^L}^+])$
|        |        |  then $D'_i = I'$
|        |        |  else $(I_1, I_2)$=split$(I')$
|        |        |        $D = D_1 \times \cdots I_1 \times \cdots \times D_n$
|        |        |        $I''$=pushleft$(f, D, i)$
|        |        |        if $I'' \neq \emptyset$
|        |        |        |  then $D'_i = [I''^L, I'^R]$
|        |        |        |  else $D = D_1 \times \cdots \times I_2 \times \cdots \times D_n$
|        |        |        |        $D'_i$ =pushleft$(f, D, i)$
|        |        |        fi
|        |        fi
|        fi
fi
```

FIGURE 3. Pushleft function: find the first leftmost solution

The previous function can be extended to perform box-consistency of a constraint. Such a function is equivalent to CI when considering one constraint C and the set of reduction functions $\{IBC_i$ for $C|$ for all i in the scheme of $C\}$. However, we won't consider it, since we want to keep the most control as we can within the general CI algorithm.

4.3. Limited local splitting

The idea consists in restricting the local splitting when reduction is not sufficient: if box-consistency is computationally expensive for a constraint C with respect to i in the scheme s, then it may be "cheaper" to try getting box-consistency with respect to another element of the scheme, or another constraint. Box-consistency of C with respect to i will then be computed later, and may become at this stage easier to compute.

To this end, we modify pushleft and pushright (see Figure 4), and add a stopping criterion based on the ratio between the width[7] of I and the width of

[7] The width of I is denoted $w(I)$ and is equal to $I^R - I^L$.

I' the result of $Newton^\star(x_i, f, I)$. If this ratio is less than the *minimum ratio* r (a data of the algorithm), then local splitting is stopped.

```
limited_pushleft
input $f : \mathbb{R}^n \to \mathbb{R};\ D_1 \times \cdots \times D_n : \mathcal{I}^n;\ r :\in [0,1]$
output $D'_i : \mathcal{I}$

$I := D_i$
if $0 \notin F_{X_i}(I)$
|  then $D'_i = \emptyset$
|  else $I' = Newton^\star(x_i, f, I)$
|       if $I' = \emptyset$
|       |  then $D'_i = \emptyset$
|       |  else if $0 \in F_{X_i}([I'^L, I'^{L^+}])$ or $w(I)/w(I') - 1 <$ r
|       |       |  then $D'_i = I'$
|       |       |  else $(I_1, I_2)$=split($I'$)
|       |       |       $D = D_1 \times \cdots I_1 \times \cdots \times D_n$
|       |       |       $I''$=limited_pushleft($f, D, i, r$)
|       |       |       if $I'' \neq \emptyset$
|       |       |       |  then $D'_i = [I''^L, I'^R]$
|       |       |       |  else $D = D_1 \times \cdots \times I_2 \times \cdots \times D_n$
|       |       |       |       $D'_i$ =limited_pushleft($f, D, i, r$)
|       |       |       fi
|       |       fi
|       fi
fi
```

FIGURE 4. Find the first leftmost root if splitting is efficient

The function limited_pushright is obtained similarly.

Limited_i_Box_Consistency is similar to i_Box_Consistency but makes calls to limited_pushleft and right.

PROPERTY 4.8. *Let $\langle D_1, \cdots, D_n; \mathcal{C}\rangle$ be a CSP, $f(\vec{x}) = 0$ a constraint of $\mathcal{C}$ with scheme s, and i an element of s. The function $LIBC_i^r$ is defined as follows:*

$$LIBC_i^r(D_1 \times \cdots \times D_n) = D_1 \times \cdots \times D_{i-1} \times D'_i \times D_{i+1} \times \cdots \times D_n$$

where $D'_i = Limited\_i\_Box\_Consistency(f, D_i, i, r)$. Then, the function $LIBC_i^r$ is a monotonic domain reduction function for the constraint $f(\vec{x}) = 0$ with scheme s on DO_s. The function $LIBC_i^r$ is not idempotent.

We obtain a collection of sets $\mathcal{LIBC}^r$ of monotonic domain reduction functions depending on r:

$$\mathcal{LIBC}^r = \{LIBC_i^r \text{ for } C | \text{ for all } C \text{ in } \mathcal{C} \text{ with scheme } s, \text{ and for all } \text{ in } s\}$$

Limited local splitting reduces the computational complexity inherent to box-consistency, but does not lead to box-consistent constraints (except when $r = 0$; in this case $LIBC_i^0 = IBC_i$). However, these functions are "stronger" than the ones based on Newton alone.

When a function $LIBC_i^r$ for C is applied after $N_i^\star$ for C, then $LIBC_i^r$ has no effect: $Newton^\star$ (which is idempotent) has already reduced the domain, and, a new application has no effect. Hence, the ratio is 0, and the local splitting is stopped. For this case, we consider the set $\mathcal{LIBC}'$ of functions $LIBC'^r_i$ where the splitting (in limited_pushleft) is performed before any computation of $Newton^\star$.

4.4. Properties of reduction functions

For a given CSP $\langle\mathcal{D};\mathcal{C}\rangle$, we thus have three sets of domain reduction functions $\mathcal{IBC}, \mathcal{N}$, and $\mathcal{N}^\star$, plus some other sets of domain reduction functions depending on r, namely $\mathcal{LIBC}^r$. We can compare (with respect to tightening of domain) elements of these sets. Let C be a constraint with scheme s. Then, for all i of s, all $I \in DO_s$, and all $r, s \in [0,1]$ such that $r < s$:

$$IBC_i(I) \subseteq LIBC_i^r(I) \subseteq LIBC_i^s(I) \subseteq N_i^\star(I) \subseteq N_i(I)$$

When $f(I) \subseteq g(I)$, we say that f is *stronger* than g (or g is *weaker* than f). In our case, the stronger domain reduction functions are, the more computationally complex and expensive are their computation.

The CSPs and the related domain reduction functions are now fed in the CI algorithm to compute the reduced CSP. Since all the functions previously described are monotonic, and the domains are finite, the CI algorithm always terminates when using (some of) these functions. Only $\mathcal{IBC}$ guarantees that box consistency of the CSP is reached. The other functions do not enforce any property, but they speed-up the computation by reducing the problem before computing box-consistency.

5. Experimental results

5.1. Implementation

We have designed a prototype of our solver for real constraints. Since we were more interested in the design of strategies, and in testing several domain reduction functions, we used Maple [7] as an implementation language. Hence, the performance is not really good compared to Numerica [23] and Declic [3], but we can easily and quickly compare reduction functions, and strategies for their application.

We tested several domain reduction functions, based on different techniques (Taylor, Newton, ...) but we focus here on the functions we presented in Section 4, and on the problem of getting CSPs that are box-consistent. From such CSPs, solutions are derived by a branch and prune technique [22]: CSPs are split and box consistency is enforced on each sub-CSPs till domains are small enough.

Internally, constraints are stored as binary trees. Using commutative properties of the arithmetic operators, we raise variables as high as possible in the trees. Hence, when initializing a set of constraints, we can already evaluate all the sub-terms that do not contain any variables.

Then, when calling a domain reduction function, all remaining variables except the one whose domain will be reduced (this variable is the highest in the tree) are replaced by their domains. Once again, sub-terms are evaluated. We thus obtain a pre-evaluated constraint[8] which is already simplified and will be used several times inside the reduction function. When reducing a CSP, this two-level pre-evaluation mechanism improves the computation of 25 percent. That is one of the reasons that lead us to chose distributed interval extension instead of natural interval extension (for which pre-evaluation is less effective since we cannot transform the constraints in a single way) although this last generally gives sharper bounds.

5.2. Strategies

We made some small changes to the selection and update functions of the `CI` algorithm (see Section 3).

1- The CII algorithm [2] is a specialized version of the CI algorithm for idempotent domain reduction functions. However, we cannot use it since not all functions are not idempotent. Thus, we specialize the update procedure as follows: if g is idempotent, g is not added to G after it has been applied.

2- Let $F = F_1 \cup F_2$ be a set of domain reduction functions such that each function of F_2 is weaker than a function of F_1[9]. Then:

$$CI(CSP, F_1 \cup F_2) = CI(CI(CSP, F_2), F_1)$$

We use this property in the CI algorithm. First, CI selects the functions in F_2 (the weaker functions). When there are no more functions from F_2 in G, functions of F_1 are chosen, and the functions from F_2 are not added to G during updates. This improves the average performance of the algorithm although few examples were solved faster using the standard selection procedure.

3- When $\mathcal{IBC}$ and a set $\mathcal{LIBC}^r$ are fed into F, we change the condition of the update procedure. When $g \in \mathcal{LIBC}^r$ has been applied to reduce D_i into D'_i, G is updated only if the reduction ratio $D_i/D'_i - 1$ is greater than r. However, we replace D_i by the new domain D'_i. The result of the CI algorithm remains

[8] A pre-evaluated constraint is a uni-variate constraint which is a sum of "products". Each product is either an interval, a variable, or an interval multiplied by a variable at a given power.

[9] This is always the case when F_1 and F_2 are elements, or union of elements of $\{\mathcal{IBC}, \mathcal{LIBC}^{r_1}, \dots, \mathcal{LIBC}^{r_n}, \mathcal{N}^\star, \mathcal{N}\}$.

unchanged, since the functions in $\mathcal{LIBC}^r$ are chosen first and that there is always a stronger function in $\mathcal{IBC}$.

4- In the selection procedure, when several functions of the same type are candidates, we choose the one that reduces the domain of the variable which occurs most often in $\mathcal{C}$.

5.3. Results

Before stating some generalities, we give some results based on particular examples (see Appendix A for the descriptions of the examples): Broyden Banded Function [9] (bb), and interval arithmetic benchmarks [10] (i1, i2, i3, i4).

In Table 5, Var represents the number of variables in the example, Dom is the initial domain of the variables, and S_i represents the set of domain reduction functions:

$$\begin{array}{lcl} S_1 & = & \mathcal{IBC}, \\ S_2 & = & \mathcal{LIBC}^{0.1} \cup S_1, \\ S_3 & = & \mathcal{LIBC}^{0.01} \cup S_2, \\ S_4 & = & \mathcal{LIBC}^{0.0001} \cup S_3, \\ S_5 & = & \mathcal{N}^\star \cup \mathcal{LIBC}'^{0.1} \cup \mathcal{LIBC}'^{0.01} \cup \mathcal{LIBC}'^{0.0001} \cup \mathcal{IBC}, \\ S_6 & = & \mathcal{N} \cup S_5, \\ S_7 & = & \mathcal{N} \cup \mathcal{N}^\star \cup \mathcal{LIBC}'^{0.01} \cup \mathcal{IBC}. \end{array}$$

For a given example, we compare a strategy (i.e., a set of reduction functions) with the best strategy we obtained: 1 means that it is the best strategy, and 1.5 means this strategy is 1.5 times slower than the best strategy, and "?" means that we could not solve the example because we ran out of memory.

Ex.	Var	Dom	S_1	S_2	S_3	S_4	S_5	S_6	S_7
i1	10	[-2,2]	1.5	1.58	1.6	1.63	1.05	1.09	1
i2	20	[-1,2]	1.8	1.8	1.82	1.8	1.2	1.1	1
i3	20	[-2,2]	2.2	2.1	2.2	1.3	1.15	1.1	1
i4	10	[-1,1]	1.7	1.2	1.13	1.13	1.15	1.18	1
bb	3	[-10^8,10^8]	2.9	1.12	1.2	1.17	1	1	1.3
bb	10	[-10^8,10^8]	?	1.02	1.03	1	1.08	1.12	1.04

FIGURE 5. Comparison of some strategies

We obtained some better strategies than the ones given in the table above. However, each of them was specialized for a single example, and did not improved the efficiency for the other examples.

For some examples (such as i1), the CSP is already box-consistent when using only $\mathcal{N} \cup \mathcal{N}^\star$. However, we need the functions of $\mathcal{IBC}$ to prove the box-consistency of the CSP.

Informally speaking, the weaker functions perform an "equitable" reduction of all the domains. On the other hand, functions of $\mathcal{IBC}$ reduce one domain as much as possible (till box-consistency of one constraint with respect to one variable is reached) before the new domain is propagated. Note some analogies with depth-first and breadth-first search.

On a larger set of experiments, we determined that S_7 is the strategy that gives the best results on average: computations with S_1 are (on average) 1.7 times longer than computations with S_7.

The weaker functions not only improve the efficiency, but also reduce the required memory (see bb(10) in Table 5). When we apply the functions from $\mathcal{IBC}$, the domains are already smaller. Hence, the local splitting is reduced i.e., the depth of the recursion in pushleft and right is reduced. Thus, the weaker functions enabled us solving examples that we could not solve (due to memory problems) using only $\mathcal{IBC}$.

6. Conclusion

We presented constraint propagation over real numbers using the general framework of chaotic iteration. We also proposed to introduce cheap domain reduction functions to preliminary reduce CSPs before enforcing box-consistency. A prototype and some experimental results confirmed that this method speeds up box consistency computation and reduces the required memory without adding any new mathematical machinery.

The main difference between our technique and Numerica [23] is that we use a more general framework and that we reduce CSPs before enforcing box consistency. Another advantage, is that some specialized functions (for example for linear/quadratic equations) can naturally be added to speed up the computation, without modifying our framework. Numerica was implemented in C and is much faster than our prototype implementation written in Maple. We expect that our techniques can increase the efficiency of Numerica when implemented in C.

Since no preliminary modification of CSPs is required for our method, it can be combined with symbolic rewriting [4] or with other solvers [17, 18, 16] without affecting the collaboration.

We plan to add some more specialized reduction functions for speeding up computation. We also envisage to use several interval extensions as in [22]. We also plan to refine the selection procedure in the chaotic iteration algorithm by choosing not only the type of function, but also the domain of the variable that will be reduced. We plan to combine this with the usual techniques of constraint propagation (such as selecting the variable with the smallest/largest domain). Finally, we plan to design a distributed version of the algorithm that will enable simultaneous reductions, storage of domain information computed during the local splitting in (limited) box-consistency functions, and simultaneous search of interval boundaries (simultaneous push left and right).

References

1. G. Alefeld and J. Herzberger. *Introduction to interval computation.* Series in Computer Science and Applied Mathematics. Academic Press, 1983.
2. K. R. Apt. From Chaotic Iteration to Constraint Propagation. In *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 36–55. Springer Verlag, 1997.
3. F. Benhamou, F. Goualard, and L. Granvilliers. Programming with the Declic Language. In *Proceedings of the International Workshop on Interval Constraints*, pages 1–13, Port Jefferson, USA, 1997.
4. F. Benhamou and L. Granvilliers. Combining local consistency, symbolic rewriting, and interval methods. In *Proceedings of AISMC3*, Steyr (Austria), 1996.
5. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Proceedings of ILPS'94, Ithaca, NY, USA*, 1994.
6. F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, July 1997.
7. K. Geddes, G. Gonnet, and B. Leong. *Maple V : Language reference manual.* Springer Verlag, New York, Berlin, Paris, 1991.
8. E. Hansen. *Global optimization using interval analysis.* Number 165 in Series in Pure and Applied Mathematics. Marcel Dekker, New York - Basel - Hong Kong, 1992.
9. E. R. Hansen and R. I. Greenberg. An Interval Newton Method. *Applied Mathematical Computing*, 12, 1983.
10. H. Hong and V. Stahl. Safe starting regions by fixed points and tightening. *Computing*, 53(3-4), 1994.
11. E. Hyvönen. Constraint reasonning based on interval arithmetic: the tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.
12. IEEE. IEEE standard for binary floating-point arithmetic. IEEE Std 754-1985, 1985. Reaffirmed 1990.
13. O. Lhomme. Consistency Techniques for Numeric CSPs. In R. Bajcsy, editor, *Proceedings of the 13th IJCAI, Chambéry, France.* IEEE Computer Society Press, 1993.
14. O. Lhomme and M. Rueher. Application des techniques CSP au raisonnement sur les intervalles. *RIA*, 11(3):283–312, 1997.
15. A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
16. P. Marti and M. Rueher. A Distributed Cooperating Constraints Solving System. *International Journal on Artificial Intelligence Tools*, 4(1&2):93–113, 1995.
17. E. Monfroy. *Collaboration de solveurs pour la programmation logique à contraintes.* PhD Thesis, Université Henri Poincaré-Nancy I, 1996. (also available in english).
18. E. Monfroy, M. Rusinowitch, and R. Schott. Implementing Non-Linear Constraints with Cooperative Solvers. In *Proc. of SAC'96*, pages 63–72,

Feb. 1996.

19. R. E. Moore. *Interval Analysis.* Series in Automatic Computation. Prentice Hall, Englewood Cliffs, N. J., 1966.
20. D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1:85–118, 1996.
21. V. Telerman and D. Ushakov. Subdefinite models as a variety of constraint programming. In *Proceedings of ICTAI'96*, Toulouse (France), 1996.
22. P. Van Hentenryck, D. McAllester, and D . Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2), 1997.
23. P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: a modeling language for global optimization.* The MIT Press, 1997.

A. Description of the examples

1. Broyden Banded Function (bb): it consistes in finding the zeros of the functions

$$f_i(x_1, \cdots, x_n) = x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j) \quad (1 \leq i \leq n)$$

where $J_i = \{j \mid j \neq i \ \wedge \ max(1, i-5) \leq j \leq min(n, i+1)\}$, and n, the number of variables, is a parameter. For each i in $[1, n]$, $D_{x_i} = [-10^8, 10^8]$.

2. i1:

$$\begin{array}{lcl}
x_1 - 0.25428722 - 0.18324757\, x_4\, x_3\, x_9 & = & 0 \\
x_2 - 0.37842197 - 0.16275449\, x_1\, x_{10}\, x_6 & = & 0 \\
x_3 - 0.27162577 - 0.16955071\, x_1\, x_2\, x_{10} & = & 0 \\
x_4 - 0.19807914 - 0.15585316\, x_7\, x_1\, x_6 & = & 0 \\
x_5 - 0.44166728 - 0.19950920\, x_7\, x_6\, x_3 & = & 0 \\
x_6 - 0.14654113 - 0.18922793\, x_8\, x_5\, x_{10} & = & 0 \\
x_7 - 0.42937161 - 0.21180486\, x_2\, x_5\, x_8 & = & 0 \\
x_8 - 0.07056438 - 0.17081208\, x_1\, x_7\, x_6 & = & 0 \\
x_9 - 0.34504906 - 0.19612740\, x_{10}\, x_6\, x_8 & = & 0 \\
x_{10} - 0.42651102 - 0.21466544\, x_4\, x_8\, x_1 & = & 0
\end{array}$$

For each i in $[1, 10]$, $D_{x_i} = [-2, 2]$.

3. i2:

$$\begin{array}{lcl}
x_1 - 0.24863995 - 0.19594124\, x_7\, x_{10}\, x_{16} & = & 0\\
x_2 - 0.87528587 - 0.05612619\, x_{18}\, x_8\, x_{11} & = & 0\\
x_3 - 0.23939835 - 0.20177810\, x_{10}\, x_7\, x_{11} & = & 0\\
x_4 - 0.47620128 - 0.16497518\, x_{12}\, x_{15}\, x_1 & = & 0\\
x_5 - 0.24711044 - 0.20198178\, x_8\, x_9\, x_{16} & = & 0\\
x_6 - 0.33565227 - 0.15724045\, x_{16}\, x_{18}\, x_{11} & = & 0\\
x_7 - 0.13128974 - 0.12384342\, x_{12}\, x_{13}\, x_{15} & = & 0\\
x_8 - 0.45937304 - 0.18180253\, x_{19}\, x_{15}\, x_{18} & = & 0\\
x_9 - 0.46896600 - 0.21241045\, x_{13}\, x_2\, x_{17} & = & 0\\
x_{10} - 0.57596835 - 0.16522613\, x_{12}\, x_9\, x_{13} & = & 0\\
x_{11} - 0.56896263 - 0.17221383\, x_{16}\, x_{17}\, x_8 & = & 0\\
x_{12} - 0.70561393 - 0.23556251\, x_{14}\, x_{11}\, x_4 & = & 0\\
x_{13} - 0.59642512 - 0.24475135\, x_7\, x_{16}\, x_{20} & = & 0\\
x_{14} - 0.46588640 - 0.21790395\, x_{13}\, x_3\, x_{10} & = & 0\\
x_{15} - 0.10607114 - 0.20920602\, x_1\, x_9\, x_{10} & = & 0\\
x_{16} - 0.26516898 - 0.21037773\, x_4\, x_{19}\, x_9 & = & 0\\
x_{17} - 0.20436664 - 0.19838792\, x_{20}\, x_{10}\, x_{13} & = & 0\\
x_{18} - 0.56003141 - 0.18114505\, x_6\, x_{13}\, x_8 & = & 0\\
x_{19} - 0.92894617 - 0.04417537\, x_7\, x_{13}\, x_{16} & = & 0\\
x_{20} - 0.57001682 - 0.17949149\, x_1\, x_3\, x_{11} & = & 0
\end{array}$$

For each i in $[1, 20]$, $D_{x_i} = [-1, 2]$.

4. i3: the constraints of i3 are the same as the ones of i2. Only the initial domains change: for each i in $[1, 20]$, $D_{x_i} = [-2, 2]$.

5. i4:

$$\begin{array}{lcl}
x_1^2 - 0.25428722 - 0.18324757\, x_4^2\, x_3^2\, x_9^2 & = & 0\\
x_2^2 - 0.37842197 - 0.16275449\, x_1^2\, x_{10}^2\, x_6^2 & = & 0\\
x_3^2 - 0.27162577 - 0.16955071\, x_1^2\, x_2^2\, x_{10}^2 & = & 0\\
x_4^2 - 0.19807914 - 0.15585316\, x_7^2\, x_1^2\, x_6^2 & = & 0\\
x_5^2 - 0.44166728 - 0.19950920\, x_7^2\, x_6^2\, x_3^2 & = & 0\\
x_6^2 - 0.14654113 - 0.18922793\, x_8^2\, x_5^2\, x_{10}^2 & = & 0\\
x_7^2 - 0.42937161 - 0.21180486\, x_2^2\, x_5^2\, x_8^2 & = & 0\\
x_8^2 - 0.07056438 - 0.17081208\, x_1^2\, x_7^2\, x_6^2 & = & 0\\
x_9^2 - 0.34504906 - 0.19612740\, x_{10}^2\, x_6^2\, x_8^2 & = & 0\\
x_{10}^2 - 0.42651102 - 0.21466544\, x_4^2\, x_8^2\, x_1^2 & = & 0
\end{array}$$

For each i in $[1, 20]$, $D_{x_i} = [-1, 1]$.