A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation

Xuan-Ha Vu, Djamila Sam-Haroud, and Boi V. Faltings

Artificial Intelligence Laboratory,
Swiss Federal Institute of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
{xuan-ha.vu, jamila.sam, boi.faltings}@epfl.ch
http://liawww.epfl.ch

Technical Report No. IC/2004/39 (April 21, 2004)

Abstract. This paper proposes a novel generic scheme enabling the combination of multiple inclusion representations to propagate numerical constraints. The scheme allows bringing into the constraint propagation framework the strength of inclusion techniques coming from different areas such as interval arithmetic, affine arithmetic or mathematical programming. The scheme is based on the DAG representation of the constraint system. This enables devising fine-grained combination strategies involving any factorable constraint system. The paper presents several possible combination strategies for creating practical instances of the generic scheme. The experiments reported on a particular instance using interval propagation, interval arithmetic, affine arithmetic and linear programming illustrate the flexibility and efficiency of the approach.

1 Introduction

Many real world applications involve the solving of problems modeled as *numerical constraints* on variables with *continuous* domains. In practice, numerical constraints can be equalities or inequalities of arbitrary type, usually expressed in *factorable* form, that is, they can be represented by elementary functions such as $+, -, \times, \div$, log, exp, sin, cos, ... Recently, many solution techniques have been proposed in *constraint programming* to solve such constraint systems. Some of them are based on *interval (constraint) propagation* and *interval arithmetic* (e.g. the works in [1], [2] and references therein), while some of them are based on *linear relaxation* and *linear programming* ([3], [4]). There have also been techniques ([5], [6]) that use *G interval* or *affine arithmetic* to solve equation systems. Most of the solution techniques are interleaved with a *bisection* search (or similar) to solve the problems exhaustively. Lately, there have been some advanced search techniques ([7], [8]) that improve the search performance for problems with non-isolated solutions (e.g., inequalities) while maintaining the same performance

for problems with isolated solutions (e.g., equalities). In general, different techniques have different strengths that are complementary. Therefore, combining the strength of different solution techniques is the subject of intensive research efforts (see [2] and references therein).

Our contributions will be described in Section 3 and Section 4. In Section 3, we propose a novel generic scheme which allows devising new combination strategies for numerical constraint propagation in a flexible way. The scheme enables the propagation to be performed using different inclusion representations on a *directed acyclic graph* (DAG) that represents the problem. Consequently, the scheme is applicable to any factorable constraint system. The goal is to provide a combination scheme that is efficient and flexible but still general enough to bring the strength of different solution techniques coming from different areas (e.g., constraint programming and mathematical programming) into the framework of constraint propagation. In order to illustrate the flexibility and efficiency of the proposed scheme, in Section 4 we propose improvements on affine arithmetic and then devise from the scheme several new combination strategies which are based on emerging techniques, namely interval constraint propagation, interval arithmetic, affine arithmetic, and linear programming. In Section 5, our experiments show that the devised technique is superior than the recent interval propagation methods in performance and quality. It even outperforms some very recent techniques in mathematical programming and constraint programming which are specially designed to solve certain constraint systems. The conclusion and future directions are given in Section 6.

2 Background and Definitions

2.1 Interval Arithmetic

When using an interval $[a, b] \subseteq \mathbb{R}$ to represent a quantity x, we mean that for each value assigned to x, the following holds

$$a \leq x \leq b \tag{1}$$

There have been different variants of the interval notion that are still valid for the techniques proposed in this paper, however, for simplicity we use the term "interval" to refer to the notion defined by (1).

Interval arithmetic is an arithmetic defined on the set of intervals, rather than the set of real numbers. According to a survey paper [9], a form of interval arithmetic perhaps first appeared in 1924 in [10], then later in [11]. Modern development of interval arithmetic began with R. E. Moore's dissertation [12] and R. E. Moore's book [13]. Another good introduction to interval arithmetic can be found in [14]. Interval arithmetic has been being used to solve numerical problems with guaranteed accuracy. Fundamentally, if $\mathbf{x}$ and $\mathbf{y}$ are two real intervals, then the four elementary operations for *idealized interval arithmetic* obey the rule

$$\mathbf{x} \diamond \mathbf{y} = \{x \diamond y \mid x \in \mathbf{x}, y \in \mathbf{y}\}, \forall \diamond \in \{+, -, \times, \div\} \tag{2}$$

Thus, the ranges of the four elementary interval arithmetic operations are exactly the ranges of the their real-valued counterparts. Although the rule (2) characterizes these operations mathematically, interval arithmetic's usefulness is due to the *operational definitions* based on interval bounds [15]. For example, if $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$, we then have:

$$\begin{aligned}
\mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
\mathbf{x} - \mathbf{y} &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
\mathbf{x} \times \mathbf{y} &= [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}] \\
1/\mathbf{x} &= [1/\overline{x}, 1/\underline{x}], \text{ if } \underline{x} > 0 \text{ or } \overline{x} < 0 \\
\mathbf{x} \div \mathbf{y} &= \mathbf{x} \times 1/\mathbf{y}
\end{aligned}$$

Moreover, if such operations are composed, *bounds on the ranges* of real functions can be obtained. For example, if given the function $f(x) = x(x-1)$, then bounds on the ranges of f over $[0, 1]$ can be obtained from its *natural extension* to interval arithmetic, i.e. $\mathbf{f}(\mathbf{x}) = \mathbf{x}(\mathbf{x}-1)$, precisely by $\mathbf{f}([0,1]) = [0,1]([0,1]-1) = [0,1][-1,0] = [-1,0]$ which contains the exact range $[-0.25, 0]$.

The finite nature of computers precludes an exact representation of the *reals*. In practice, the real set, $\mathbb{R}$, is approximated by a finite set $\mathbb{F}_\infty = \mathbb{F} \cup \{-\infty, +\infty\}$, where $\mathbb{F}$ is the set of floating-point numbers. The set of real intervals is then approximated by the set, $\mathbb{I}$, of intervals with bounds in $\mathbb{F}_\infty$. The power of interval arithmetic lies in its implementation on computers. In particular, *outwardly rounded* interval arithmetic allows *rigorous enclosures* for the ranges of operations and functions. This makes a qualitative difference in scientific computations, since the results are now intervals in which the exact result must lie. Interval arithmetic can be carried out for virtually any expression that can be evaluated with floating-point arithmetic. However, since interval arithmetic is only *subdistributive*, expressions that are equivalent in real arithmetic differ in interval arithmetic. Therefore, computations should be arranged so that overestimation of ranges is minimized. Readers are referred to [15] and [2] for more details on basic interval methods.

2.2 Affine Arithmetic

Affine arithmetic [16] is an extension of interval arithmetic which keeps track of correlations between computed and input quantities. Therefore, it is resistant to the catastrophic loss of accuracy often observed in long-running interval computations. In particular, a real quantity x is represented by an *affine form* which is a first-degree polynomial of the form

$$x = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n \tag{3}$$

where $x_0, \ldots, x_n$ are real coefficients and $\varepsilon_1, \ldots, \varepsilon_n$ are (noise) variables (also called *noise symbols*) taking values in $[-1, 1]$.

Similar to interval arithmetic, affine arithmetic also allows to use rounded floating-point arithmetic to construct *rigorous enclosures* for the ranges of operations and functions [17]. In affine arithmetic, affine operations such as $\alpha x + \beta y + \gamma$

are obtained exactly, except the rounding errors, by the following formula:

$$\alpha x + \beta y + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n} (\alpha x_i + \beta y_i)\varepsilon_i \tag{4}$$

However, non-affine operations can only be computed by approximations. In general, the exact result of a non-affine operation has form $f^*(\varepsilon_1, \ldots, \varepsilon_n)$, where f^* is a non-linear function. In practice, this result is then approximated by an affine function $f^a(\varepsilon_1, \ldots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \ldots + z_n\varepsilon_n$. A new term $z_k\varepsilon_k$ is used to represent the difference between f^* and f^a, hence, the result has the affine form

$$z = z_0 + z_1\varepsilon_1 + \ldots + z_n\varepsilon_n + z_k\varepsilon_k \tag{5}$$

where the maximum absolute error $z_k \geq \sup\{|f^*(\varepsilon_1, \ldots, \varepsilon_n) - f^a(\varepsilon_1, \ldots, \varepsilon_n)| : \forall(\varepsilon_1, \ldots, \varepsilon_n) \in [-1, 1]^k\}$. An important goal is to keep this maximum absolute error as small as possible (see the *Chebyshev Approximation Theory*).

Ranges obtained with affine arithmetic may be substantially more accurate than those obtained with interval arithmetic. However, the operations of affine arithmetic are often more expensive than those of interval arithmetic. Some comparisons on interval and affine arithmetic methods can be found in [17], [18] and [19].

2.3 Directed Acyclic Graph

Hereafter, we recall some fundamental concepts in graph theory related to the representation of a constraint system [20].

Definition 1 (Directed Multigraph). *A* directed multigraph (V, E, f) *consists of a finite set of* vertices *(also called* nodes*)* V*, a finite set of* edges E *and a mapping* $f : E \to V \times V$ *such that* $\forall e \in E : f_s(e) \neq f_t(e)$*, where* $f = (f_s, f_t)$*.*

Definition 2 (Directed Multigraph with Ordered Edges). *A* directed multigraph with ordered edges *is a quadruple* $(V, E, f, \preceq)$ *such that* (V, E, f) *is a directed multigraph and* $(E, \preceq)$ *is a totally ordered set.*

Definition 3 (Directed Path). *Let* $\mathcal{G} = (V, E, f)$ *be a directed multigraph. A* directed path *from* $v_1 \in V$ *to* $v_2 \in V$ *is a sequence,* $\{e_i\}_{i=1}^n$*, of edges such that* $v_1 = f_s(e_1)$*,* $v_2 = f_t(e_n)$*, and* $\forall i \in \{1, \ldots, n-1\} : f_t(e_i) = f_s(e_{i+1})$*. The directed path is called a* cycle *if* $v_1 = v_2$*.* $\mathcal{G}$ *is called* acyclic *if it does not contain a cycle.*

Definition 4. *Let* (V, E, f) *be a directed multigraph. For any two vertices* $v_1, v_2 \in V$ *we say that* v_1 *is a* parent *of* v_2 *and* v_2 *is a* child *of* v_1 *if* $\exists e \in E : f_s(e) = v_2 \wedge f_t(e) = v_1$*. We call* v_1 *an* ancestor *of* v_2 *and* v_2 *a* descendant *of* v_1 *if there exists a directed path from* v_2 *to* v_1*.*

Theorem 1. *For every directed acyclic multigraph* (V, E, f) *there exists a total order* $\preceq$ *on vertices* V *such that* $\forall v \in V$ *: if* u *is an ancestor of* v*, then* $v \preceq u$*.*

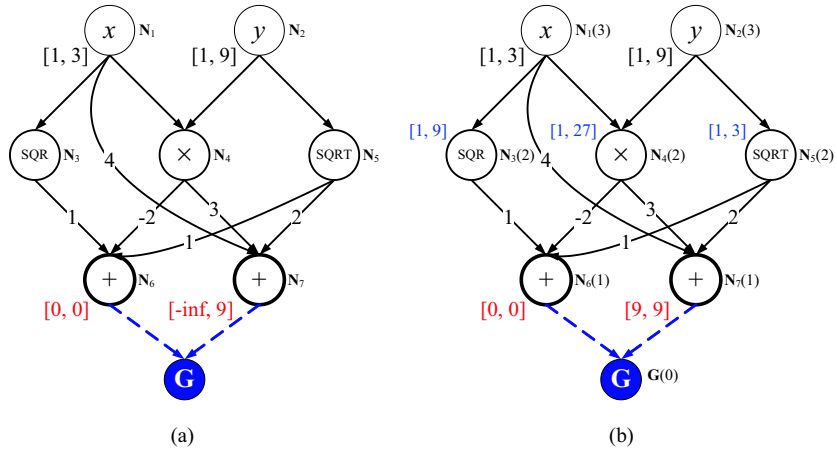

Fig. 1. The DAG representation (a) before and (b) after interval evaluations

Following the approach in [20], we use a directed acyclic multigraph, whose edges are totally ordered and whose vertices are ordered by an order in Theorem 1, to represent a constraint system, we therefore call it a *Directed Acyclic Graph* (DAG). In a DAG, every node represents an elementary operation such as $+$, $\times$, $\div$, log, exp, ... and every edge represents the computational flow associated with a coefficient. The ordering of edges is needed for non-communicative operations like the division, and not really necessary for communicative operations. For convenience, a virtual ground node is added to a DAG to be the parent of all the nodes representing the constraints. The reason that we use multigraphs instead of simple graphs for the representation is the fact that some n-ary operations can take the same input more than once, for example, when the expression x^x is represented by the operation x^y.

Example 1. The DAG representation of the following constraint system

$$\begin{cases} x^2 - 2xy + \sqrt{y} = 0, \\ 4x + 3xy + 2\sqrt{y} \leq 9 \\ x \in [1,3],\ y \in [1,9] \end{cases}$$

is given in Figure 1. $\{\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \mathbf{N}_4, \mathbf{N}_5, \mathbf{N}_6, \mathbf{N}_7\}$ is an ordering of the nodes satisfying Theorem 1.

3 Combination Scheme for Constraint Propagation

3.1 Generalization of Inclusion Concepts

We generalize the concepts related to *inclusion function* as follows.

Definition 5 (Inclusion Representation). *Given a set $\mathcal{A}$. A couple $\mathcal{I} = (\mathcal{R}, \mu)$, where $\mathcal{R}$ is a set of* representing objects *and μ is a function from $\mathcal{R}$ to $2^{\mathcal{A}}$, is called an* inclusion representation *of $\mathcal{A}$ if there exists a surjective function $\rho : 2^{\mathcal{A}} \to \mathcal{R}$ such that $\forall S \subseteq \mathcal{A} : S \subseteq \mu(\rho(S))$. In this case, ρ is called the* representing function *of $\mathcal{I}$ and μ is called the* evaluating function *of $\mathcal{I}$.*

Let $\mathcal{I} = (\mathcal{R}, \mu)$ be an inclusion representation of $\mathbb{R}$. We call $\mathcal{I}$ a *real inclusion representation* of $\mathbb{R}$ if each representing object $T \in \mathcal{R}$ is a tuple consisting of real constants, and the evaluating function μ can be defined by

$$\mu(T) \equiv \{f_T(V_T) \mid V_T \in D_T\} \tag{6}$$

where f_T is a real-valued function (with T as a tuple of parameters) and V_T is a finite sequence of variables taking values in real domains D_T. The representation (6) is called a *real representation* of μ.

It is easy to see that the interval form (1) is a real inclusion representations of $\mathbb{R}$, where each representing object $T \in \mathcal{R}$ is a couple of reals (a, b), $V_T = (x)$, f_T is an identity function, and μ is defined by

$$\mu(T) \equiv \{x \mid x \in [a, b]\} \tag{7}$$

The affine form (3) is also a real inclusion representation of $\mathbb{R}$, where each representing object is a tuple $T = (x_0, \ldots, x_n, 1, \ldots, n)$,[1] and $V_T = (\varepsilon_1, \ldots, \varepsilon_n)$ are the variables of the linear function $f_T(\varepsilon_1, \ldots, \varepsilon_n) = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n$. Hence, the real representation of the evaluating function is defined by

$$\mu(T) \equiv \{x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n \mid (\varepsilon_1, \ldots, \varepsilon_n) \in [-1, 1]^n\} \tag{8}$$

Definition 6 (Inclusion Function). *Given two sets X, Y and a function $f : X \to Y$. Let $\mathcal{I}_X = (\mathcal{R}_X, \mu_X)$ and $\mathcal{I}_Y = (\mathcal{R}_Y, \mu_Y)$ be two inclusion representations of X and Y, respectively. A function $F : \mathcal{R}_X \to \mathcal{R}_Y$ is called an* inclusion function *of f, if $\forall S \subseteq X, \forall T \in \mathcal{R}_X : S \subseteq \mu_X(T) \Rightarrow \{f(x) \mid x \in S\} \subseteq \mu_Y(F(T))$.*

Definition 7 (Inclusion Converter). *Let $\mathcal{I}_1 = (\mathcal{R}_1, \mu_1)$ and $\mathcal{I}_2 = (\mathcal{R}_2, \mu_2)$ be two inclusion representations of the same set. A function $c : \mathcal{R}_1 \to \mathcal{R}_2$ is called an* inclusion converter from $\mathcal{I}_1$ to $\mathcal{I}_2$ *if $\forall S \in \mathcal{R}_1 : \mu_1(S) \subseteq \mu_2(c(S))$.*

Theorem 2. *Let $\mathcal{I}_X = (\mathcal{R}_X, \mu_X)$, $\mathcal{I}_Y = (\mathcal{R}_Y, \mu_Y)$ and $\mathcal{I}_Z = (\mathcal{R}_Z, \mu_Z)$ be inclusion representations of three sets X, Y and Z, respectively. If $F : \mathcal{R}_X \to \mathcal{R}_Y$ and $G : \mathcal{R}_Y \to \mathcal{R}_Z$ are inclusion functions of two functions $f : X \to Y$ and $g : Y \to Z$ respectively, then the composite function $G \circ F$ is an inclusion function of the composite function $g \circ f$.*

Corollary 1. *Let $\mathcal{I} = (\mathcal{R}, \mu)$ be an inclusion representation of $\mathbb{R}$. If all elementary operations defined on $\mathcal{R}$ are inclusion functions of their counterparts on $\mathbb{R}$, then all factorable functions built on $\mathcal{R}$ are also inclusion functions of their counterparts on $\mathbb{R}$.*

The proof of Theorem 2 follows directly from Definition 5 and Definition 6. Corollary 1 is a straightforward consequence of Theorem 2. The elementary operations in interval arithmetic and affine arithmetic are inclusion functions of their real-valued counterparts, therefore, as a result of Corollary 1, all the factorable operations/functions defined in interval arithmetic (or affine arithmetic) are also inclusion functions of their real-valued counterparts.

[1] In implementation, only non-zero coefficients and their indices should be stored.

3.2 A General Combination Scheme

In this section, we describe a general combination scheme that combines the strength of different real inclusion representations for constraint propagation. In this scheme, the input constraint system is represented by a DAG as described in Section 2.3. The data stored at each node, $\mathbf{N}$, of the DAG consist of a representing object for each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu)$ of $\mathbb{R}$ and a constrained range $\tau(\mathbf{N}) \subseteq \mathbb{R}$ often be an interval. We denote by $\mathcal{R}(\mathbf{N})$ the representing object (in $\mathcal{I}$) stored at $\mathbf{N}$.

Definition 8 (Inclusion Constraint System, ICS). *Let $(\mathcal{R}, \mu)$ be a real inclusion representation of $\mathbb{R}$ defined by* (6)*, $\mathbf{N}$ a node of a DAG representing a constraint system. The* inclusion constraint system *induced by a representing object $T \equiv \mathcal{R}(\mathbf{N})$ and a* constrained domain *$D \subseteq \mathbb{R}$ is defined by*

$$\texttt{ICS}(T, D) \equiv \begin{cases} \{var(\mathbf{N}) \in D_T, var(\mathbf{N}) \in D\}(var(\mathbf{N}) \equiv V_T) & \text{if } f_T \text{ is identity,} \\ \{f_T(V_T) = var(\mathbf{N}), V_T \in D_T, var(\mathbf{N}) \in D\} & \text{otherwise;} \end{cases}$$

where $var(\mathbf{N})$ (i.e. the representing variable of $\mathbf{N}$) and the variables in V_T are the variables of the constraint system.

From (7) and (8) we can see that constructing the inclusion constraint system induced by a representing object and a constrained range $[a, b] \subseteq \mathbb{R}$ for interval form and affine form is trivial.

Definition 9 (NEV, PCS). *Let $\mathbf{N}$ be a node of a DAG representing a constraint system, $\{\mathbf{C}_i\}_{i=1}^k$ the children of $\mathbf{N}$, $f : \mathbb{R}^k \to \mathbb{R}$ the elementary operation represented by $\mathbf{N}$, $\mathcal{S}$ a finite set of real inclusion representations. The following constraint system is called the* pruning constraint system *in $\mathcal{S}$ at $\mathbf{N}$:* $\texttt{PCS}(\mathbf{N}, \mathcal{S}) \equiv$

$$\begin{cases} \{\bigwedge_{i=1}^k \texttt{ICS}(\mathcal{R}(\mathbf{C}_i), \tau(\mathbf{C}_i))\} & \text{if } \mathbf{N} \text{ is ground,} \\ \left\{ \begin{array}{l} f(var(\mathbf{C}_1), \ldots, var(\mathbf{C}_k)) = var(\mathbf{N}) \wedge \\ \bigwedge_{(\mathcal{R},\mu)\in\mathcal{S}}(\texttt{ICS}(\mathcal{R}(\mathbf{N}), \tau(\mathbf{N})) \wedge \bigwedge_{i=1}^k \texttt{ICS}(\mathcal{R}(\mathbf{C}_i), \tau(\mathbf{C}_i))) \end{array} \right\} & \text{otherwise.} \end{cases}$$

For each $\mathcal{I} = (\mathcal{R}, \mu) \in \mathcal{S}$, let $f_{\mathcal{I}} : \mathcal{R}^k \to \mathcal{R}$ be an inclusion function of f. The following assignment is called the node evaluation *in $\mathcal{I}$ at $\mathbf{N}$ (if $\mathbf{N} \neq$ ground):*

$$\texttt{NEV}(\mathbf{N}, \mathcal{I}) \equiv \left\{ \begin{array}{l} \mathcal{R}(\mathbf{N}) := \mathcal{R}(\mathbf{N}) \cap \tau(\mathbf{N}) \cap f_{\mathcal{I}}(\mathcal{R}(\mathbf{C}_1), \ldots, \mathcal{R}(\mathbf{C}_k)); \\ \tau(\mathbf{N}) := \tau(\mathbf{N}) \cap \mu(\mathcal{R}(\mathbf{N})); \end{array} \right\}$$

Definition 10 (Pruning Technique). *A solution technique is called a* pruning technique *for a real constraint system if it is capable of reducing some domains of the variables in that system.*

Let $\mathcal{G}$ be a DAG representing the input constraint system. The following algorithm scheme, called CIRD, uses two waiting lists. The first waiting list stores the nodes waiting for evaluation, denoted by $\mathcal{L}_e$. The second waiting list stores the nodes waiting for node pruning, denoted by $\mathcal{L}_p$. Note that each node can appear once at a time in a waiting list. The set of real inclusion representations for use in the scheme is denoted by $\mathcal{E}$. Each real inclusion representation in $\mathcal{E}$ provides the elementary operations that are inclusion functions of their real-valued counterparts. The following gives the main steps of the CIRD scheme.

A Scheme for Combining Inclusion Representations on DAG (CIRD)

1. **Initialization Phase.**
   (a) **Initial Node Evaluation.** Select an algorithm for visiting DAGs in an ordering described in Theorem 1. When visiting a node $\mathbf{N} \in \mathcal{G}$, perform the node evaluation $\texttt{NEV}(\mathbf{N}, \mathcal{I})$ for each $\mathcal{I} \in \mathcal{E}$.
   (b) **Initialize Waiting Lists.** Set $\mathcal{L}_e := \emptyset$, $\mathcal{L}_p := \{$the list of all nodes representing the active constraints together with all the real inclusion representations of $\mathcal{E}\}$.
2. **Propagation Phase.** Repeat this step until both $\mathcal{L}_e$ and $\mathcal{L}_p$ become empty, or the limit on the number of iterations is reached.
   (a) **Get The Next Node.** Select a strategy for getting a node $\mathbf{N}$ (and the set $\mathcal{S}$ of real inclusion representations associated with $\mathbf{N}$ in the corresponding list) from the two waiting lists $\mathcal{L}_e$ and $\mathcal{L}_p$.
   (b) **Node Evaluation.** *Do this step only if* $\mathbf{N}$ *was taken from* $\mathcal{L}_e$ *at Step 2a.* For each $\mathcal{I} = (\mathcal{R}, \mu) \in \mathcal{S}$ do the following steps:[2]
      i. Perform the node evaluation $\texttt{NEV}(\mathbf{N}, \mathcal{I})$. If this returns an empty set, the algorithm stops with an infeasible status.
      ii. If the changes of $\mathcal{R}(\mathbf{N})$ and $\tau(\mathbf{N})$ at Step 2(b)i are considered enough to re-evaluate the parents of $\mathbf{N}$, then put each node in $parents(\mathbf{N})$ (associated with $\mathcal{I}$) into $\mathcal{L}_e$, if $\mathbf{N}$ is not the ground node, or into $\mathcal{L}_p$ otherwise.
      iii. If the changes of $\mathcal{R}(\mathbf{N})$ and $\tau(\mathbf{N})$ at Step 2(b)i are considered enough to do a node pruning at $\mathbf{N}$ again, then put $(\mathbf{N}, \mathcal{I})$ into $\mathcal{L}_p$.
   (c) **Node Pruning.** *Do this step only if* $\mathbf{N}$ *was taken from* $\mathcal{L}_p$ *at Step 2a.*
      i. Select a subset $\mathcal{T} \subseteq \mathcal{S}$ such that for each $\mathcal{I} \in \mathcal{T}$ there are efficient pruning techniques for the constraint system $\texttt{PCS}(\mathbf{N}, \mathcal{I})$.
      ii. Partition $\mathcal{T}$ into subsets such that for each subset $\mathcal{U}$ of the partition there is a pruning technique that may efficiently reduce the domains of the variables of the system (or a subsystem of) $\texttt{PCS}(\mathbf{N}, \mathcal{U})$. After that, apply the associated pruning technique to each system (or a subsystem of) $\texttt{PCS}(\mathbf{N}, \mathcal{U})$ in a certain order. If this process returns an empty set, the algorithm stops with an infeasible status.
      iii. Let $\mathcal{K}$ be the set of all the nodes whose evaluating functions in form (6) contain some variables whose domains were reduced at Step 2(c)ii. Select a subset $\mathcal{H}$ of $\mathcal{K}$, *for example*, such that each node $\mathbf{M}$ in $\mathcal{H}$ is a descendant of $\mathbf{N}$. For each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu) \in \mathcal{H}$ such that the representation of $\mu(\mathcal{R}(\mathbf{M}))$ in form (6) contains some variables whose domains were reduced at Step 2(c)ii, update $\mathcal{R}(\mathbf{M})$ using those newly reduced domains, then update $\tau(\mathbf{M}) := \tau(\mathbf{M}) \cap \mu(\mathcal{R}(\mathbf{M}))$. If empty is obtained, the algorithm stops with an infeasible status.
         A. If the changes of $\mathcal{R}(\mathbf{M})$ and $\tau(\mathbf{M})$ are considered enough to re-evaluate $\mathbf{M}$'s parents, put each node in $parents(\mathbf{M})$ associated with $\mathcal{I}$ into $\mathcal{L}_e$.
         B. If the changes of $\mathcal{R}(\mathbf{M})$ and $\tau(\mathbf{M})$ are considered enough to do a node pruning at $\mathbf{M}$, put $(\mathbf{M}, \mathcal{I})$ into $\mathcal{L}_p$.

[2] Combining several inclusion representations for better evaluation by using inclusion converters is also an option to try.

4 Specific Combination Strategies as Instances of CIRD

In the rest of the paper, we denote by $\lfloor E \rfloor$ (resp. $\lceil E \rceil$) some lower approximation (reps. some upper approximation) in $\mathbb{F}$ of the expression E such that $\lfloor E \rfloor \leq E$ (resp. $E \leq \lceil E \rceil$). We use the notion $E = \langle E \rangle \pm e$ to mean that $\langle E \rangle$ is an approximation in $\mathbb{F}$ of E, and the corresponding bound on the absolute rounding error is e, that is $\langle E \rangle - e \leq E \leq \langle E \rangle + e$. Readers are referred to [17] for some rounding techniques in floating-point arithmetic on simple elementary operations.

4.1 Modifications To Affine Arithmetic

Revised Affine Form. One of the limits of the standard affine arithmetic is that the number of noise symbols grows gradually during the computation and the computation cost heavily depends on this number. Inspired by the ideas in [18], [5] and [6], we use a revised affine form similar to (5) but the new term $z_k\varepsilon_k$ is replaced by a non-negative accumulative error $[-e_z, e_z]$ which represents the maximum absolute error z_k of non-affine operations. In other words, the *revised affine form* of a real-valued quantity $\hat{x}$ is defined by

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1, 1] \tag{9}$$

which consists of two separated parts: the standard affine part of length n, and the interval part. Where the magnitude of the accumulative error, $e_x \geq 0$, is represented by the interval part. That is, for each value x of the quantity $\hat{x}$ (say $x \in \hat{x}$), there exist $\varepsilon_x \in [-1, 1], \varepsilon_i \in [-1, 1]$ $(i = 1, \ldots, n)$ such that $x = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x\varepsilon_x$. We then say it is of length n. The affine operations are now defined by

$$\hat{z} \equiv \alpha\hat{x} + \beta\hat{y} + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n} (\alpha x_i + \beta y_i)\varepsilon_i + (|\alpha|e_x + |\beta|e_y)[-1, 1] \tag{10}$$

Therefore, during the computation the length of revised affine forms will not exceed the number of noise symbols at the beginning, i.e. the number of variables of the input constraint system. In rigorous computing, e_z will be used to accumulate the rounding errors in floating-point arithmetic, namely (10) can be interpreted as follows

$$z_0 = \langle \alpha x_0 + \beta y_0 + \gamma \rangle \pm e_0,\ z_i = \langle \alpha x_i + \beta y_i \rangle \pm e_i,\ e_z = \lceil |\alpha|e_x + |\beta|e_y + \sum_{i=0}^{n} e_i \rceil \tag{11}$$

Another limit of the standard affine form is that it is not capable of handling half-lines of the form $(-\infty, a]$ and $[a, +\infty)$, while this is important for many computation methods, especially constraint propagation and search techniques. Hence, we propose to associate each quantity $\hat{x}$ with a data field $x_\infty \in \{-1, 0, +1\}$. The revised affine form is then interpreted as follows:[3]

[3] For simplicity, we allow zero coefficients in the formulae in the paper, however in implementation one should keep only nonzero coefficients.

$$\hat{x} = \begin{cases} (-\infty, +\infty) & \text{if } e_x = +\infty, \\ (-\infty, x_0] & \text{if } x_\infty = -1, \\ [x_0, +\infty) & \text{if } x_\infty = +1, \\ x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1,1] & \text{otherwise.} \end{cases} \tag{12}$$

In an operation, if the domain of a variable is unbounded, i.e. in the first three cases of (12), the other variables are converted into interval forms for that operation performed in interval arithmetic, then the result is converted back to affine form. Therefore, in the rest of paper, we only need to discuss about the last case of (12). The set of all objects in revised affine form is denoted by $\mathbb{A}$.

Unary Operations. We give the following theorem as a basis for finding affine approximations of elementary univariate functions in a rigorous manner.

Theorem 3 (Affine Approximation of Univariate Functions). *Let f be a differentiable function on $[a,b]$, where $a < b$ in $\mathbb{R}$, and $d_\alpha(x) \equiv f(x) - \alpha x$.*

1. *(a) If $\forall x \in [a,b] : \alpha \geq f'(x)$, then $\forall x \in [a,b] : \alpha x + d_\alpha(b) \leq f(x) \leq \alpha x + d_\alpha(a)$.*
   (b) If $\forall x \in [a,b] : \alpha \leq f'(x)$, then $\forall x \in [a,b] : \alpha x + d_\alpha(a) \leq f(x) \leq \alpha x + d_\alpha(b)$.
2. *If f' is continuous and monotone increasing on $[a,b]$, we have*
   (a) $\forall \alpha \in [f'(a), f'(b)], \exists c \in [a,b] : f'(c) = \alpha$.
   (b) Let $g : \mathbb{R} \to \mathbb{R}$ be a function such that $g(\alpha) = d_\alpha(c)$, then $\forall x \in [a,b] : \alpha x + g(\alpha) \leq f(x) \leq \alpha x + \max\{d_\alpha(a), d_\alpha(b)\}$.
3. *If f' is continuous and monotone decreasing on $[a,b]$, we have*
   (a) $\forall \alpha \in [f'(b), f'(a)], \exists c \in [a,b] : f'(c) = \alpha$.
   (b) Let $g : \mathbb{R} \to \mathbb{R}$ be a function such that $g(\alpha) = d_\alpha(c)$, then $\forall x \in [a,b] : \alpha x + \min\{d_\alpha(a), d_\alpha(b)\} \leq f(x) \leq \alpha x + g(\alpha)$.

Proof. We give the proofs for the case 1a and case 2. The proofs are similar for the other cases. Considering the case 1a, we have

$f(x) - (\alpha x + d_\alpha(b)) = (f(x) - f(b)) - \alpha(x - b)$
$= f'(\xi)(x - b) - \alpha(x - b)$ | for some $\xi \in [x, b]$ (Mean Value Theorem)
$= (x - b)(f'(\xi) - \alpha) \geq 0$ | $x \leq b, f'(\xi) \leq \alpha$

Analogously, we have $f(x) - (\alpha x + d_\alpha(a)) \leq 0$. The proof of the case 1a therefore follows these results. We now prove the case 2a: because $\alpha \in [f'(b), f'(a)]$ and f' is continuous on $[a,b]$ then there exists $c \in [a,b]$ as required. Hereafter, we give the proof of the case 2b. For all $x \in [a,b]$ we have

$f(x) - (\alpha x + g(\alpha)) = (f(x) - f(c)) - \alpha(x - c)$
$= f'(\xi)(x - c) - \alpha(x - c)$ | $\xi \in [x, c]$ or $\xi \in [c, x]$ (Mean Value Theorem)
$= (x - c)(f'(\xi) - f'(c))$ | $\alpha = f'(c)$
≥ 0 | ξ is between x and c, f' is monotone increasing

Moreover, if $x \in [a, c]$, we have $f(x) - (\alpha x + d_\alpha(a)) = (f(x) - f(a)) - \alpha(x - a)$
$= f'(\eta)(x - a) - \alpha(x - a)$ | for some $\eta \in [a, x]$ (Mean Value Theorem)
$= (x - a)(f'(\eta) - f'(c))$ | $\alpha = f'(c)$
≤ 0 | $\eta \leq c$, f' is monotone increasing

Table 1. Examples of functions $f \in \mathcal{C}^1([a,b])$ satisfying the conditions of Theorem 3

$f(x)$	$[a,b]$ is a subset of	$f'(x)$	f'	$g(\alpha)$
x^2	$(-\infty,+\infty)$	$2x$	$\uparrow$	$-\alpha^2/4$
$\sqrt{x}$	$[0,+\infty)$	$1/(2\sqrt{x})$	$\downarrow$	$1/(4\alpha) : \alpha > 0$
e^x	$(-\infty,+\infty)$	e^x	$\uparrow$	$\alpha(1-\log\alpha) : \alpha > 0$
$\log x$	$(0,+\infty)$	$1/x$	$\downarrow$	$-(1+\log\alpha) : \alpha > 0$
$x^n : n \geq 2$ is even	$(-\infty,+\infty)$	nx^{n-1}	$\uparrow$	$(1-n)\sqrt[n-1]{(\alpha/n)^n}$
$x^n : n \geq 3$ is odd	$(-\infty,0]$	nx^{n-1}	$\downarrow$	$(n-1)\sqrt[n-1]{(\alpha/n)^n} : \alpha \geq 0$
$x^n : n \geq 3$ is odd	$[0,+\infty)$	nx^{n-1}	$\uparrow$	$(1-n)\sqrt[n-1]{(\alpha/n)^n} : \alpha \geq 0$
$1/x^n : n \geq 2$ is even	$(-\infty,0);(0,+\infty)$	$-n/x^{n+1}$	$\uparrow$	$(n+1)\sqrt[n+1]{(-\alpha/n)^n}$
$1/x^n : n \geq 1$ is odd	$(-\infty,0)$	$-n/x^{n+1}$	$\downarrow$	$-(n+1)\sqrt[n+1]{(-\alpha/n)^n} : \alpha < 0$
$1/x^n : n \geq 1$ is odd	$(0,+\infty)$	$-n/x^{n+1}$	$\uparrow$	$(n+1)\sqrt[n+1]{(-\alpha/n)^n} : \alpha < 0$
$x^r : r \notin [0,1]$	$(0,+\infty)$	rx^{r-1}	$\uparrow$	$(1-r)(\alpha/r)^{(r/(r-1))} : \alpha r > 0$
$x^r : r \in (0,1)$	$(0,+\infty)$	rx^{r-1}	$\downarrow$	$(1-r)(\alpha/r)^{(r/(r-1))} : \alpha > 0$

Similarly, if $x \in [c,b]$ we have $f(x) - (\alpha x + d_\alpha(b)) \leq 0$. Hence, we have $f(x) \leq \alpha x + \max\{d_\alpha(a), d_\alpha(b)\}$ for all $x \in [a,b]$. The proof is then completed. □

To illustrate the usefulness of Theorem 3, in Table 1 we give the functions f' and g for some elementary functions, and in Figure 2 we give a procedure to find a safe Chebyshev affine approximation of a function $f \in \mathcal{C}^1([a,b])$ such that f' is monotone, when given the function g satisfying the conditions in Theorem 3.

Proposition 1. *Let $\alpha\hat{x} + \beta + \delta[-1,1]$ be the revised affine form produced by the procedure in Figure 2, where $[a,b]$ is the range of $\hat{x}$. Suppose that $f \in \mathcal{C}^1([u,v])$ and f' is monotone on $[u,v]$, where $[u,v] \supseteq [a,b]$ such that $f'(v) \geq \lceil f'(b) \rceil$ if f' is monotone increasing, or $f'(u) \geq \lceil f'(a) \rceil$ if f' is monotone decreasing. We have $\forall x \in \hat{x} : f(x) \in \alpha\hat{x} + \beta + \delta[-1,1]$.*

Proof. Following the Mean Value Theorem, there is some $c^* \in [a,b]$ such that

$$\alpha^* = f'(c^*) = (f(b) - f(a))/(b-a)$$

Therefore

$$\begin{aligned} & \alpha^* \leq \lceil (\lceil f(b) \rceil - \lfloor f(a) \rfloor)/(b-a) \rceil = \alpha \\ \Rightarrow\ & \alpha \geq \min\{f'(a), f'(b)\} \qquad (f' \text{ is monotone}) \end{aligned}$$

Hereafter, we prove for the case f' is monotone increasing, the proof for the other case is similar. If $\alpha > \lceil f'(b) \rceil$, then $\forall x \in [a,b] : \alpha > f'(x)$. Hence, following the case $1a$ of Theorem 3, we have, for all $x \in [a,b]$

$$\begin{aligned} & \alpha x + d_\alpha(b) \leq f(x) \leq \alpha x + d_\alpha(a) \\ \Rightarrow\ & \alpha x + d_{min} \leq \alpha x + d_\alpha(b) \leq f(x) \leq \alpha x + d_\alpha(a) \leq \alpha x + d_{max} \end{aligned}$$

If $\alpha \leq f'(b)$, the result follows directly from the case 2 of Theorem 3. In the rest of the proof, we consider the case $f'(b) < \alpha \leq \lceil f'(b) \rceil$. Similar to the above, we have $\alpha x + d_\alpha(b) \leq f(x) \leq \alpha x + d_\alpha(a)$. Moreover, applying the case 2 of Theorem 3 to $[b,v]$, we have

$$\begin{aligned} & x \in [b,v] : \alpha x + g(\alpha) \leq f(x) \\ \Leftrightarrow\ & x \in [b,v] : g(\alpha) \leq f(x) - \alpha x \\ \Rightarrow\ & g(\alpha) \leq f(b) - \alpha b = d_\alpha(b) \end{aligned}$$

procedure `AffineApproximation`(**in** : $\hat{x}, f \in \mathcal{C}^1([a,b]), f', g$; **out** : $\alpha\hat{x} + \beta + \delta[-1,1]$)
  $f_a := \lfloor f(a) \rfloor$; $f_b := \lceil f(b) \rceil$; $\alpha := \lceil (f_b - f_a)/(b-a) \rceil$;
  if f' is monotone increasing on $[a,b]$ **then**
    $d_a := \lceil f(a) \rceil - \lfloor \alpha a \rfloor$;
    if $\alpha > \lceil f'(b) \rceil$ **then**
      $d_{min} := \lfloor f(b) \rfloor - \lceil \alpha b \rceil$; $d_{max} := d_a$;
    else
      $d_{min} := \lfloor g(\alpha) \rfloor$; $d_{max} := \max\{d_a, f_b - \lfloor \alpha b \rfloor\}$;
    end-if
  else-if f' is monotone decreasing on $[a,b]$ **then**
    $d_b := \lfloor f(b) \rfloor - \lceil \alpha b \rceil$;
    if $\alpha > \lceil f'(a) \rceil$ **then**
      $d_{min} := d_b$; $d_{max} := \lceil f(a) \rceil - \lfloor \alpha a \rfloor$;
    else
      $d_{min} := \min\{f_a - \lceil \alpha a \rceil, d_b\}$; $d_{max} := \lceil g(\alpha) \rceil$;
    end-if
  end-if
  $\beta := midpoint([d_{min}, d_{max}])$; $\delta := radius([d_{min}, d_{max}])$;
end

Fig. 2. This is a procedure to find a Chebyshev affine approximation of a function $f \in \mathcal{C}^1([a,b])$ such that f' is monotone, when given the function g in Theorem 3

Therefore, for all $x \in [a,b]$ we have

$$\alpha x + g(\alpha) \le f(x) \le \alpha x + d_\alpha(a) = \max\{\alpha x + d_\alpha(a), \alpha x + d_\alpha(b)\}$$
$$\Rightarrow \alpha x + d_{min} \le f(x) \le \alpha x + d_{max}$$

As a result, $\forall x \in \hat{x} : f(x) \in \alpha\hat{x} + \beta + \delta[-1,1]$. The proof is then completed. □

Readers are referred to Section 2 of [21] for a method to compute affine approximations of non-differentiable functions.

Multiplication. Similar to the product of two G intervals in [5] and [6], the product of two revised affine forms $\hat{x}$ and $\hat{y}$ of length n is another revised affine form $\hat{z}$ of length n defined as follows

$$z_0 = x_0 y_0 + 0.5 \sum_{i=1}^{n} x_i y_i, \;\; z_i = x_0 y_i + y_0 x_i \;\; (i = 1, \ldots, n) \tag{13}$$

$$e_z = e_x e_y + e_y \sum_{i=0}^{n} |x_i| + e_x \sum_{i=0}^{n} |y_i| + \sum_{i=1}^{n} |x_i| \sum_{i=1}^{n} |y_i| - 0.5 \sum_{i=1}^{n} |x_i y_i| \tag{14}$$

This is similar to, but tighter than, the formula for multiplication in [6] when exactly porting into revised affine form. The time complexity of (10) and (14) is $O(n)$. In rigorous computing, we use the following computations.

$$u = \lceil \sum_{i=1}^{n} |x_i| \rceil, \;\; v = \lceil \sum_{i=1}^{n} |y_i| \rceil \tag{15}$$

$$z_0 = \langle x_0y_0 + 0.5\sum_{i=1}^{n} x_iy_i\rangle \pm e_0,\ z_i = \langle x_0y_i + y_0x_i\rangle \pm e_i\ (i = 1,\ldots,n) \quad (16)$$

$$e_z = \lceil e_xe_y + e_y(|x_0| + u) + e_x(|y_0| + v) + uv + \sum_{i=0}^{n} e_i\rceil - \lfloor 0.5\sum_{i=1}^{n} |x_iy_i|\rfloor \quad (17)$$

Proposition 2. *The affine multiplication function defined by* $\{(13),(14)\}$ *or by* $\{(15),(16),(17)\}$ *is an inclusion function of the real multiplication.*

Proof. It suffices to prove that $\forall x \in \hat{x}, y \in \hat{y} : xy \in \hat{z} = z_0 + \sum_{i=1}^{n} z_i\varepsilon_i + e_z[-1,1]$. By definition, there exist $e_x, e_y \in [-1,1]$ such that

$$x = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x\varepsilon_x,\ y = y_0 + \sum_{i=1}^{n} y_i\varepsilon_i + e_y\varepsilon_y$$

Therefore, we have

$$\begin{aligned}
xy &= x_0y_0 + \textstyle\sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i + x_0e_y\varepsilon_y + y_0e_x\varepsilon_x + e_xe_y\varepsilon_x\varepsilon_y + \\
&\quad e_y\textstyle\sum_{i=1}^{n} x_i\varepsilon_y\varepsilon_i + e_x\sum_{i=1}^{n} y_i\varepsilon_x\varepsilon_i + \sum_{i,j=1;i\neq j}^{n} x_iy_j\varepsilon_i\varepsilon_i + \sum_{i=1}^{n} x_iy_i\varepsilon_i^2 \\
&\in x_0y_0 + \textstyle\sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i + |x_0|e_y[-1,1] + |y_0|e_x[-1,1] + e_xe_y[-1,1] + \\
&\quad e_y\textstyle\sum_{i=1}^{n} |x_i|[-1,1] + e_x\sum_{i=1}^{n} |y_i|[-1,1] + \sum_{i,j=1;i\neq j}^{n} |x_iy_j| + \\
&\quad 0.5\textstyle\sum_{i=1}^{n}(x_iy_i + |x_iy_i|[-1,1]) \\
&= (x_0y_0 + 0.5\textstyle\sum_{i=1}^{n} x_iy_i) + \sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i + [-1,1]\times \\
&\quad (e_xe_y + e_y\textstyle\sum_{i=0}^{n} |x_i| + e_x\sum_{i=0}^{n} |y_i| + \sum_{i=1}^{n} |x_i|\sum_{i=1}^{n} |y_i| - 0.5\sum_{i=1}^{n} |x_iy_i|) \\
&\subseteq z_0 + \textstyle\sum_{i=1}^{n} z_i\varepsilon_i + e_z[-1,1] = \hat{z}\ \text{(Follow } \{(13),(14)\} \text{ or } \{(15),(16),(17)\}).
\end{aligned}$$

□

Division. In our implementation, we compute the quotient $\hat{z} = \hat{x}/\hat{y}$ by rewriting it as $\hat{x} \times (1/\hat{y})$. However, it is worth mention that in [6] and [22], the authors have proposed better methods for division that have some interesting properties such as $\hat{x}/\hat{x} = 1$.

4.2 New Combination Strategies for Constraint Propagation

In the rest, we will abuse the notions $\mathbb{I}$ and $\mathbb{A}$ to denote the real inclusion representations, $(\mathbb{I}, \mu_{\mathbb{I}})$ and $(\mathbb{A}, \mu_{\mathbb{A}})$, defined on interval arithmetic and revised affine arithmetic, respectively; where the function $\mu_{\mathbb{I}}$ is defined by (7) and the function $\mu_{\mathbb{A}}$ follows (12). In general, the performance of a propagator following the `CIRD` scheme depends on the design of each step in the scheme. In this section, we propose some simple strategies for each step in the `CIRD` scheme using the two inclusion representations, $\mathbb{I}$ and $\mathbb{A}$. Combining different strategies at all the steps makes different combination strategies for constraint propagation.

Step 1a: Initial Node Evaluation. A post-order visiting or a recursive evaluation starting from the ground node is an option for the visit at Step 1a.

procedure NodeLevel(**in** : *DAG*)
  Initialize the level of each node to zero.
  for each visit in pre-order going from a parent **P** to its children **N** **do**
    $level(\mathbf{N}) := \max\{level(\mathbf{N}), level(\mathbf{P}) + 1\}$;
end

Fig. 3. This is a procedure assigning a node level to each node in a DAG.

Step 2a: Get The Next Node. At first, we assign a *node level* to each node in the DAG representing the constraint system such that each ancestor has a lower level than that of their descendants, hence, an ordering in Theorem 1 can be obtained easily. Figure 3 gives a simple procedure for this purpose. $\mathcal{L}_p$ is sorted in the ascending order of node levels. It is to maintain that ancestors being taken into pruning processes before their descendants. $\mathcal{L}_e$ is sorted in the descending order of node levels. It is to sure that descendants being evaluated before their ancestors. There are two simple strategies to get the next node from $\{\mathcal{L}_e, \mathcal{L}_p\}$. The first one is to get the next node from $\mathcal{L}_p$ whenever it is not empty. The second one is to get the next node from one of the two waiting lists until it becomes empty, then switch to the other list. In our implementation, we use the first simple strategy. More complicated strategies for choosing the next node can be used as alternatives, for example, based on the pruning efficiency of nodes.

Step 2b: Node Evaluation. For the node evaluation at each node **N**, we can perform $\texttt{NEV}(\mathbf{N}, \mathbb{A})$ and $\texttt{NEV}(\mathbf{N}, \mathbb{I})$ in any order, if **N** is not the ground node. At Step 2(b)ii, Step 2(b)iii and Step 2(c)iii, we only count on the changes of $\tau(\mathbf{N})$ in our current implementation. A change of $\tau(\mathbf{N})$ is often considered enough if the ratio of the new width to the old width is less than a number predefined $r_w \in (0, 1)$ and the difference between the old width and the new width is greater than a predefined number $d_w > 0$ (see [1] for details). More complicated criteria that have been used in constraint programming can be used as alternatives.

Step 2c: Node Pruning. The subset $\mathcal{T}$ at this step can be chosen as $\{\mathbb{I}, \mathbb{A}\}$. For node pruning, we use $\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\})$ and the following subsystem of $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$:

$$\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\}) \equiv \left\{ \begin{array}{l} f(var(\mathbf{C}_1), \ldots, var(\mathbf{C}_k)) = var(\mathbf{N}); \\ var(\mathbf{N}) \in \tau(\mathbf{N}); \\ \bigwedge_{i=1}^{k} (var(\mathbf{C}_i) \in \tau(\mathbf{C}_i)); \end{array} \right\} \text{if } \mathbf{N} \text{ is not ground.}$$

$$\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\}) \equiv \left\{ \begin{array}{ll} \{\bigwedge_{i=1}^{k} \texttt{ICS}(\mathbb{A}(\mathbf{C}_i), \tau(\mathbf{C}_i))\} & \text{if } \mathbf{N} \text{ is ground,} \\ \{\texttt{ICS}(\mathbb{A}(\mathbf{N}), \tau(\mathbf{N})) \wedge \bigwedge_{i=1}^{k} \texttt{ICS}(\mathbb{A}(\mathbf{C}_i), \tau(\mathbf{C}_i))\} & \text{otherwise,} \end{array} \right.$$

$$\text{where } \texttt{ICS}(\mathbb{A}(\mathbf{M}), D) \equiv \left\{ \begin{array}{l} x_{\mathbf{M},0} + \sum_{i=1}^{k} x_{\mathbf{M},i} \varepsilon_i + e_{\mathbf{M}} \varepsilon_{\mathbf{M}} = var(\mathbf{M}); \\ \varepsilon_i \in [-1, 1] \ (i = 1, \ldots, n); \\ \varepsilon_{\mathbf{M}} \in [-1, 1]; var(\mathbf{M}) \in D; \end{array} \right\}$$

Example 2. We give the node levels for the example described in Section 2.3 in brackets next to the node names in Figure 1*b*. At the beginning, we have

$\tau(\mathbf{N}_1) = \mathbb{I}(\mathbf{N}_1) = [1,3]; \quad \mathbb{A}(\mathbf{N}_1) = 2 + \varepsilon_1$
$\tau(\mathbf{N}_2) = \mathbb{I}(\mathbf{N}_2) = [1,9]; \quad \mathbb{A}(\mathbf{N}_2) = 5 + 4\varepsilon_2$
$\tau(\mathbf{N}_i) = \mathbb{I}(\mathbf{N}_i) = \mathbb{A}(\mathbf{N}_i) = [-\infty, +\infty]$ $(i = 3, 4, 5)$
$\tau(\mathbf{N}_6) = \mathbb{I}(\mathbf{N}_6) = [0,0]; \quad \mathbb{A}(\mathbf{N}_6) = 0$
$\tau(\mathbf{N}_7) = \mathbb{I}(\mathbf{N}_7) = [-\infty, 9]; \quad \mathbb{A}(\mathbf{N}_7) = [-\infty, 9]$

After the initial node evaluation, we have the following changes:

$\tau(\mathbf{N}_3) = \mathbb{I}(\mathbf{N}_3) = [1,9]; \quad \mathbb{A}(\mathbf{N}_3) = 4.5 + 4\varepsilon_1 + 0.5[-1,1]$
$\tau(\mathbf{N}_4) = \mathbb{I}(\mathbf{N}_4) = [1,27]; \quad \mathbb{A}(\mathbf{N}_4) = 10 + 5\varepsilon_1 + 8\varepsilon_2 + 4[-1,1]$
$\tau(\mathbf{N}_5) = \mathbb{I}(\mathbf{N}_5) = [1,3]; \quad \mathbb{A}(\mathbf{N}_5) = 2.125 + \varepsilon_2 + 0.125[-1,1]$
$\tau(\mathbf{N}_6) = \mathbb{I}(\mathbf{N}_6) = [0,0]; \quad \mathbb{A}(\mathbf{N}_6) = -13.375 - 6\varepsilon_1 - 15\varepsilon_2 + 8.625[-1,1]$
$\tau(\mathbf{N}_7) = \mathbb{I}(\mathbf{N}_7) = [9,9]; \quad \mathbb{A}(\mathbf{N}_7) = 42.25 + 19\varepsilon_1 + 26\varepsilon_2 + 12.25[-1,1]$

Denoting the variable $var(\mathbf{N}_i)$ by v_i, we have, for example,

$$\mathtt{PCS}(\mathbf{N}_6, \{\mathbb{A}\}) \equiv \mathtt{PCS}(\mathbf{N}_6, \{\mathbb{I}\}) \wedge \mathtt{PCS}_L(\mathbf{N}_6, \{\mathbb{A}\})$$

where

$$\mathtt{PCS}(\mathbf{N}_6, \{\mathbb{I}\}) \equiv \begin{cases} v_3 - 2v_4 + v_5 = v_6 \\ v_3 \in [1,9];\ v_4 \in [1,27];\ v_5 \in [1,3];\ v_6 \in [0,0] \end{cases}$$

and

$$\mathtt{PCS}_L(\mathbf{N}_6, \{\mathbb{A}\}) \equiv \begin{cases} 4.5 + 4\varepsilon_1 + 0.5\varepsilon_{\mathbf{N}_3} = v_3 \\ 10 + 5\varepsilon_1 + 8\varepsilon_2 + 4\varepsilon_{\mathbf{N}_4} = v_4 \\ 2.125 + \varepsilon_2 + 0.125\varepsilon_{\mathbf{N}_5} = v_5 \\ -13.375 - 6\varepsilon_1 - 15\varepsilon_2 + 8.625\varepsilon_{\mathbf{N}_6} = v_6 \\ (\varepsilon_1, \varepsilon_2, \varepsilon_{\mathbf{N}_3}, \varepsilon_{\mathbf{N}_4}, \varepsilon_{\mathbf{N}_5}, \varepsilon_{\mathbf{N}_6}) \in [-1,1]^6 \\ v_3 \in [1,9];\ v_4 \in [1,27];\ v_5 \in [1,3];\ v_6 \in [0,0] \end{cases}$$

Backward Propagation. If $\mathbf{N}$ is not the ground, the domains of the variables of the constraint system $\mathtt{PCS}(\mathbf{N}, \{\mathbb{I}\})$ can be pruned by a pruning technique which is called *backward propagation* in [1] and [20]. In brief, let f be the elementary operation represented by a node $\mathbf{N}$, we then have the relation $var(\mathbf{N}) = f(\{var(\mathbf{C}_i)\}_{i=1}^k)$. For each i in $\{1, \ldots, k\}$, the backward propagation computes a cheap evaluation of the i-th projection of the relation $var(\mathbf{N}) = f(\{var(\mathbf{C}_i)\}_{i=1}^k)$ onto $var(\mathbf{C}_i)$. In case there exist a function $g_i : \mathbb{R}^k \to \mathbb{R}$ such that we can write $var(\mathbf{C}_i) = g_i(var(\mathbf{N}), \{var(\mathbf{C}_j)\}_{j=1;j\neq i}^k)$. Let G_i be an inclusion function of g_i in $\mathbb{I}$. In this case, the backward propagation at $\mathbf{N}$ is defined by

$$\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap G_i(\mathbb{I}(\mathbf{N}), \{\mathbb{I}(\mathbf{C}_j)\}_{j=1;j\neq i}^k) \ (i = 1, \ldots, k) \tag{18}$$

For instance, in Example 2 we have $v_6 = f(v_3, v_4, v_5)$ with $f(x,y,z) = x - 2y + z$. Hence, we can write $v_3 = g_1(v_6, v_4, v_5),\ v_4 = g_2(v_6, v_3, v_5),\ v_5 = g_3(v_6, v_3, v_4)$; where $g_1(x,y,z) = x + 2y - z,\ g_2(x,y,z) = (-x + y + z)/2,\ g_3(x,y,z) = x - y + 2z$.

In another case often seen in practice, f is a univariate function (i.e. $k = 1$), we can use the backward propagation defined by $\mathbb{I}(\mathbf{C}_1) := \mathbb{I}(\mathbf{C}_1) \cap f^{-1}(\mathbb{I}(\mathbf{N}))$, where $f^{-1}(.)$ denotes the reciprocal image.

After the backward propagation, at Step 2(c)iii we only need to consider k nodes $\mathcal{H} = \{\mathbf{C}_i \mid i = 1, \ldots, k\}$ for update and for putting into the waiting lists.

Affine Pruning. In $\mathbb{A}$, each variable of the input constraint system is associated with one noise symbol ε_i $(i = 1, \ldots, n)$. The system $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$ is a linear constraint system, therefore, the domains of the variables of $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$ can be pruned by using a *safe* linear programming technique [23]. If the operation represented by $\mathbf{N}$ is linear, we can apply a *safe* linear programming technique to $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$, instead of $\texttt{PCS}_L(\mathbf{N}, \{\mathbb{A}\})$, to get tighter bounds on the variables. For efficiency, only the domains of the variables $\{var(\mathbf{C}_i)\}_{i=1}^k$ and/or $\{\varepsilon_i\}_{i=1}^n$ are needed to be pruned. We can devise three possible pruning strategies for Step 2(c)iii. The first strategy only requires to prune the domains of $\{var(\mathbf{C}_i)\}_{i=1}^k$, after that, considers the update for $\mathcal{H} = \{\mathbf{C}_i\}_{i=1}^k$. The second strategy only requires to prune the domains of $\{\varepsilon_i\}_{i=1}^n$. The third strategy is to prune the domains of both $\{var(\mathbf{C}_i)\}_{i=1}^k$ and $\{\varepsilon_i\}_{i=1}^n$. For the last two strategies, the set $\mathcal{H}$ can be chosen as any subset of the set of $\mathbf{N}$'s descendants whose noise variables in $\mu_{\mathbb{A}}$ have just been pruned. In our implementation, we use the second pruning strategy with two options for $\mathcal{H}$: the set of $\mathbf{N}$'s descendants or the set of variables associated with ε_i $(i = 1, \ldots, n)$. If for each $i \in \{1, \ldots, n\}$ the new domain of noise variable ε_i is $[a_i, b_i] \subseteq [-1, 1]$, then the range update at $\mathbf{M} \in \mathcal{H}$ will be

$$\tau(\mathbf{M}) := \tau(\mathbf{M}) \cap (x_{\mathbf{M},0} + \sum_{i=1}^{n} x_{\mathbf{M},i}[a_i, b_i] + e_{\mathbf{M}}[-1, 1]) \tag{19}$$

The cost of linear programming is expensive, therefore, we should use the affine pruning technique only if the pruning ratio is high. We propose to use the affine pruning technique only if the accumulative error $e_{\mathbf{M}}$ of each node $\mathbf{M}$ involving the above linear systems is small enough, that is, the range of the operation at $\mathbf{M}$ lies in a thin slot between two hyperplanes $x_{\mathbf{M},0} + \sum_{i=1}^{n} x_{\mathbf{M},i}\varepsilon_i - e_{\mathbf{M}}$ and $x_{\mathbf{M},0} + \sum_{i=1}^{n} x_{\mathbf{M},i}\varepsilon_i + e_{\mathbf{M}}$ in the space of the noise variables $(\varepsilon_1, \ldots, \varepsilon_n)$.

The combination of backward propagation and affine pruning techniques makes different strategies for node pruning in the `CIRD` scheme.

5 Experiments

Comparison with Linearization-based Techniques. The first technique to compare with is a recent mathematical technique, `A2`, in [6] which was specially designed to solve non-linear equation systems. It converts the equation system into *separable form* then uses affine arithmetic to enclose the system by a linear relaxation system $\{\mathbf{L}(x, y) = Ax + By + b, x \in \mathbf{x}, y \in \mathbf{y}\}$, where A and B are real matrices, and b is a real vector. This technique has to assume a posterior-condition that A is invertible. No rigorous rounding technique is found in [6]. We take the example used for illustrating the power of `A2` in [6] for the comparison:

$$\begin{cases} ((4x_3 + 3x_6)x_3 + 2x_5)x_3 + x_4 = 0, \\ ((4x_2 + 3x_6)x_2 + 2x_5)x_2 + x_4 = 0 \\ ((4x_1 + 3x_6)x_1 + 2x_5)x_1 + x_4 = 0, \\ x_4 + x_5 + x_6 = 0 \\ (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_2 + (((x_3 + x_6)x_3 + x_5)x_3 + x_4)x_3 = 0 \\ (((x_1 + x_6)x_1 + x_5)x_1 + x_4)x_1 + (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_3 = 0 \\ x_1 \in [0.0333, 0.2173],\ x_2 \in [0.4000, 0.6000],\ x_3 \in [0.7826, 0.9666] \\ x_4 \in [-0.3071, -0.1071],\ x_5 \in [1.1071, 1.3071],\ x_6 \in [-2.1000, -1.9000] \end{cases} \tag{20}$$

Table 2. Comparison between `Quad` and `CIRD1`: time is in seconds; $\#It$ is the number of iterations; $\#Box$ is the number of boxes in output. The cells are filled with "n/a" if results are not yet available for comparison in this submission, due to our limited access to the code of `Quad`.

	`Quad`				`CIRD1`				Time Ratio
Problem	$\#It$	$\#Box$	Time (s)	CPU speed	$\#It$	$\#Box$	Time (s)	CPU speed	`Quad`/`CIRD1`
Gough-Steward	24	4	183.0	1.0 GHz	912	4	9.9	800 MHz	23.1
Yama196 $[n = 30]$	108	16	31.4	2.66 GHz	25	2	8.5	2.0 GHz	4.9
Yama196 $[n = 60]$	n/a	n/a	n/a	n/a	18	2	50.0	2.0 GHz	
Yama196 $[n = 100]$	n/a	n/a	n/a	n/a	20	2	194.8	2.0 GHz	
Yama196 $[n = 200]$	n/a	n/a	n/a	n/a	20	2	1305.5	2.0 GHz	
Yama196 $[n = 300]$	n/a	n/a	n/a	n/a	20	2	4580.6	2.0 GHz	

The system (20) is known to have an unique solution. To solve (20) at the resolution 10^{-5} using a bisection search, `A2` has to perform 917 iterations in 3.46 seconds on a 1.7 GHz Pentium PC to reduce the problem to 5 boxes (see [6]); while an instance of our scheme, called `CIRD1`, has to perform 54 iterations in only 0.25 seconds on a 2.0 GHz Pentium PC to reduce the problem to 3 boxes. Hence, `CIRD1` is about 11.7 times faster than `A2` for the system (20), while it is more rigorous and accurate than `A2`.

Another technique to compare with is a very recent filtering technique called `Quad` in [3], which was specifically designed to address quadratic constraints and an extension of `Quad` in [4]. We take two problems, *Gough-Steward* and *Yama196*, from [3] and [4] respectively for comparison. *Gough-Steward* is a 9-dimensional quadratic equation system in Robotics having four solutions [3]. There is probably a typo mistake at the third term in the fourth equation of *Gough-Steward* in [3], that should be corrected to z_2z_3 as follows (otherwise, the problem will have no solution).

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 = 31 \\ x_2^2 + y_2^2 + z_2^2 = 39 \\ x_3^2 + y_3^2 + z_3^2 = 29 \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 = 51 \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 = 50 \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 = 34 \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 = -32 \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 = 8 \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 = 20 \\ x_1 \in [-2.00, 5.57],\ y_1 \in [-5.57, 2.70],\ z_1 \in [0.00, 5.57] \\ x_2 \in [-6.25, 1.30],\ y_2 \in [-6.25, 2.70],\ z_2 \in [-2.00, 6.25] \\ x_3 \in [-5.39, 0.70],\ y_3 \in [-5.39, 3.11],\ z_3 \in [-3.61, 5.39] \end{cases} \tag{21}$$

Yama196 is a series of high dimensional problems consisting of n variables and n equations of form $\{(n+1)^2x_{i-1} - 2(n+1)^2x_i + (n+1)^2x_{i+1} + e^{x_i} = 0,\ x_i \in [-10, 10] \mid i = 1, \ldots, n\}$, where $x_0 = x_{n+1} = 0$. Similar to [4], we use the resolution 10^{-8} for these problems. Table 2 gives the comparison between `CIRD1` and `Quad` (at the same resolution) for the above two problems whose results has been reported in [3] and [4].

Comparison with Interval Propagation Techniques. We have carried out experiments on `CIRD1` and two other recent interval propagation techniques. The

first one is the Box Consistency in ILOG Solver 6.0, denoted by `BOX`. The second one is called `HC4` (Revised Hull Consistency) in [1]. The experiments are carried out on 33 problems which are unbiasedly selected and divided into 5 test cases. The test case T_1 consists of 7 problems with isolated solutions that are solvable by all three propagators. The test case T_2 consists of 5 problems with isolated solutions that are solvable by only two propagators. The test case T_3 consists of 8 problems with isolated solutions that cause at least two of three techniques being stopped due to timeout or due to running more than 10^6 iterations. The test case T_4 consists of 7 small problems with continuum of solutions that are solvable at resolution 10^{-2}. The test case T_5 consists of 6 hard problems with continuum of solutions that are solvable at resolution 10^{-1}. The timeout value is 6000 seconds for all the test cases, it will be used as the running time for the techniques which is timeout in the next result analysis (i.e. we are in favor of slow techniques). For the first three test cases, the resolution is 10^{-4} and the search to be used is bisection. For the last two test cases, the search to be used is a simple search technique, called `UCA6`, for inequalities (see [7, 8]). The comparison of the interval propagation techniques is based on the measures of

1. *The running time:* The relative ratio of the running time of each propagator to that of `CIRD1` is called the *relative time ratio.*
2. *The number of boxes:* The relative ratio of that number of boxes in the output of each propagator to that of `CIRD1` is called the *relative cluster ratio.*
3. *The number of iterations:* the number of iterations in search needed to solve the problems. The relative ratio of the number of iterations used by each propagator to that of `CIRD1` is called the *relative iteration ratio.*
4. *The volume of boxes (only for T_1, T_2, T_3):* We consider the reduction per dimension when replacing the set of output boxes by a volume-equivalent hypercube. The relative ratio of the reduction gained by each propagator to that of `CIRD1` is called the *relative reduction ratio.*
5. *The volume of inner boxes (only for T_4, T_5):* The ratio of the volume of inner boxes to the volume of all output boxes is called the *inner volume ratio.*

The overviews of results in our experiments are given in Table 3 and Table 4. Clearly, `CIRD1` is superior than `BOX` and `HC4` in performance and quality for the problems with isolated solutions. `CIRD1` still outperforms the others for the problems with continuum of solutions while being a little better than the others in quality of the output.

6 Conclusion

In this paper, we propose a novel generic scheme, `CIRD`, for constraint propagation using different inclusion representations on DAG. The scheme is applicable to most of known inclusion representations, including interval arithmetic, affine arithmetic, polyhedral/quadratic enclosures and their generalizations. The modifications and improvements on the rigorous computations of affine arithmetic are also proposed. As a result, we give several new combination strategies for

Table 3. (a) The average of the relative time ratios is taken over all the problems in the test cases T_1, T_2, T_3; the averages of the other relative ratios are taken over the problems in the test case T_1, i.e. over the problems which are solvable by all the techniques. (b) The averages of the relative ratios are taken over all the problems in the test cases T_4, T_5. In general, the lower the relative ratio, the better the performance/quality; and the higher the inner volume ratio, the better the quality.

	(a) *Isolated Solutions*				(b) *Continuum of Solutions*			
Propagator	Relative time ratio	Relative reduction ratio	Relative cluster ratio	Relative iteration ratio	Relative time ratio	Inner volume ratio	Relative cluster ratio	Relative iteration ratio
`CIRD1`	1.000	1.000	1.000	1.000	1.000	0.945	1.000	1.000
`BOX`	1687.162	6.014	10.057	3.769	3.414	0.944	1.102	1.056
`HC4`	3791.425	9.102	33.157	5.019	60.101	0.941	1.168	1.118

Table 4. This table contains the averages of the relative time ratios taken over the problems in each test case.

	(a) *Isolated Solutions*			(b) *Continuum of Solutions*	
Propagator	Test case T_1	Test case T_2	Test case T_3	Test case T_4	Test case T_5
`CIRD1`	1.00	1.00	1.00	1.00	1.00
`BOX`	8.35	4848.89	204.25	2.33	4.68
`HC4`	19.30	11088.74	266.23	31.42	93.56

constraint propagation based on interval arithmetic, affine arithmetic, interval constraint propagation and (safe) linear programming. After all, we show by experiments that one implementation, `CIRD1`, outperforms recent techniques by 1-4 orders of magnitude in speed, while still being better in quality measures.

A direction for near future is to try all the proposed strategies and options in Section 4.2 to investigate different combination strategies using interval arithmetic and affine arithmetic under different settings. Another direction for future is to use the quadratic form/arithmetic in [18] and use quadratic programming techniques (e.g. [3]) in the way similar to the above affine pruning technique. Moreover, developing learning strategies for choosing the next node from the waiting lists based on the pruning efficiency of nodes is another possible way to improve the overall performance of the propagation.

7 Acknowledgments

This research is funded by the European Commission and the Swiss Federal Education and Science Office through the COCONUT project (IST-2000-26063). We would like to thank ILOG for the licenses of ILOG Solver/CPLEX used in the COCONUT project.

References

1. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising Hull and Box Consistency. In: Proceedings of the International Conference on Logic Programming (ICLP'99), Las Cruces, USA (1999) 230–244

2. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. First edn. Springer (2001)
3. Lebbah, Y., Rueher, M., Michel, C.: A Global Filtering Algorithm for Handling Systems of Quadratic Equations and Inequations. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'2003). Volume LNCS 2470., Springer (2003) 109–123
4. Lebbah, Y., Michel, C., Rueher, M.: Global Filtering Algorithms Based on Linear Relaxations . In: Notes of the 2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS 2003), Switzerland (2003)
5. Kolev, L.: Automatic Computation of a Linear Interval Enclosure. Realiable Computing **7** (2001) 17–18
6. Kolev, L.: An Improved Interval Linearization for Solving Non-Linear Problems. In: 10th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN2002), France (2002)
7. Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Search Techniques for Non-linear CSPs with Inequalities. In: Proceedings of the 14th Canadian Conference on Artificial Intelligence. (2001)
8. Vu, X.H., Sam-Haroud, D., Silaghi, M.C.: Numerical Constraint Satisfaction Problems with Non-isolated Solutions. In: Global Optimization and Constraint Satisfaction. Volume LNCS 2861., Springer-Verlag (2003) 194–210
9. Kearfott, B.: Interval Computations: Introduction, Uses, and Resources. Euromath Bulletin **2(1)** (1996) 95–112
10. Burkill, J.: Functions of Intervals. Proceedings of the London Mathematical Society **22** (1924) 375–446
11. Sunaga, T.: Theory of an Interval Algebra and its Applications to Numerical Analysis. RAAG Memoirs **2** (1958) 29–46
12. Moore, R.: Interval Arithmetic and Automatic Error Analysis in Digital Computing. PhD thesis, Stanford University, USA (1962)
13. Moore, R.: Interval Analysis. Prentice Hall, Englewood Cliffs, NJ (1966)
14. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic Press, New York, NY (1983)
15. Hickey, T., Ju, Q., Van Emden, M.: Interval Arithmetic: from Principles to Implementation. Journal of the ACM (JACM) **48(5)** (2001) 1038–1068
16. Comba, J., Stolfi, J.: Affine Arithmetic and its Applications to Computer Graphics. In: Proceedings of SIBGRAPI'93, Brazil (1993)
17. Stolfi, J., de Figueiredo, L.: Self-Validated Numerical Methods and Applications. In: Monograph for 21st Brazilian Mathematics Colloquium (IMPA), Brazil (1997)
18. Messine, F.: Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization. Journal of Universal Computer Science **8(11)** (2002) 992–1015
19. Martin, R., Shou, H., Voiculescu, I., Bowyer, A., Wang, G.: Comparison of Interval Methods for Plotting Algebraic Curves. Computer Aided Geometric Design **19(7)** (2002) 553–587
20. Schichl, H., Neumaier, A.: Interval Analysis on Directed Acyclic Graphs for Global Optimization (2004) preprint - University of Vienna, Autria.
21. Kolev, L.: A New Method for Global Solution of Systems of Non-Linear Equations. Realiable Computing **4** (1998) 125–146
22. Miyajima, S., Miyata, T., Kashiwagi, M.: A New Dividing Method in Affine Arithmetic. IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences **E86-A(9)** (2003) 2192–2196
23. Neumaier, A., Shcherbina, O.: Safe Bounds in Linear and Mixed-Integer Programming. Mathematical Programming A **99** (2004) 283–296