Combining Multiple Inclusion Representations in Numerical Constraint Propagation

Xuan-Ha Vu, Djamila Sam-Haroud and Boi Faltings
Artificial Intelligence Laboratory,
Swiss Federal Institute of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
http://liawww.epfl.ch
{xuan-ha.vu,jamila.sam,boi.faltings}@epfl.ch

***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 and 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 constraint propagation, interval arithmetic, affine arithmetic and linear programming illustrate the flexibility and efficiency of the approach.**

I. Introduction

Many real world applications involve solving constraint satisfaction problems with *continuous domains*, called numerical constraint satisfaction problems (NSCPs). In practice, numerical constraints can be equalities or inequalities of arbitrary type, usually expressed in *factorable* form (that is, they can be recursively composed of elementary operations such as $+$, $-$, $\times$, $\div$, $\log$, $\exp$, $\sin$, $\cos$, $\ldots$). In other words, such an NCSP can be expressed as follows

$$F(x) \in \mathbf{b},\ x \in \mathbf{x}, \tag{1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$ is a factorable function, x is a vector of n real variables, $\mathbf{x}$ and $\mathbf{b}$ are interval vectors of sizes n and m respectively.

Many solution techniques have been proposed in *constraint programming* to solve numerical constraint systems. Some of them are based on *interval constraint propagation* and *interval arithmetic* (some among them are [1], [2], [3], [4] and references therein), while others rely on *linear relaxation* and *linear programming* [5], [6]. There have also been mathematical techniques [7], [8] that use *G interval* and *affine arithmetic* to solve equation systems. Most of the solution techniques are interleaved with a *bisection* search to solve the problems exhaustively. Lately, there have been some advanced search techniques [9], [10] 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 many intensive research efforts (see [4] and references therein).

Our contributions will be described in the sections III, IV, V and VI. At first, in Section III we generalize inclusion concepts in order to present different inclusion techniques in a common framework that makes it possible to insert most of inclusion techniques into the scheme proposed in this paper. In Section IV, we propose some modifications to affine arithmetic to make it efficient for the computations proposed in this paper. In Section V, 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 virtually 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 VI we 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 VII, our experiments show that the devised technique is superior to recent interval constraint 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 is finally given in Section VIII.

II. Background and Notation

A. Interval Arithmetic and Affine Arithmetic

When using an interval $[a, b] \subseteq \mathbb{R}$ to represent a real-valued quantity x, we mean that

$$a \leq x \leq b \tag{2}$$

Interval arithmetic is an extension of real arithmetic defined on the set of real intervals, rather than the set of real numbers.

Modern interval arithmetic was originated independently in late 1950s by several researchers; including M. WARMUS [11], T. SUNAGA [12] and R. E. MOORE [13]; with MOORE finally setting the firm foundation for the field in his many publications. We assume that readers are familiar with interval arithmetic. Otherwise, we would recommend [14], [15], [4], [16] for more details on interval arithmetic and basic interval methods.

Affine arithmetic [17] is an extension of interval arithmetic which keeps track of correlations between computed and input quantities. In particular, a real-valued quantity x is represented by an *affine form* defined as follows

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

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

Similarly to interval arithmetic, affine arithmetic also allows using rounded floating-point arithmetic to construct *rigorous enclosures* for the ranges of operations and functions [18]. In affine arithmetic, affine operations such as $\alpha x + \beta y + \gamma$ ($\alpha, \beta, \gamma \in \mathbb{R}$) are obtained exactly, except the rounding errors, by the following formula:

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

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 nonlinear 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

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

where the maximum absolute error z_k satisfies

$$z_k \geq \sup\{|f^*(\varepsilon) - f^a(\varepsilon)| : \forall \varepsilon = (\varepsilon_1, \ldots, \varepsilon_n) \in [-1, 1]^n\}.$$

An important goal is to keep the maximum absolute error as small as possible. This is a subject of *Chebyshev approximation theory* which is a well-developed field with a vast literature. 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 [18], [19], [20].

B. Directed Acyclic Graph

We assume that readers are already familiar with fundamental concepts from graph theory such as *directed multigraph with ordered edges* and *directed acyclic graph/multigraph*. Otherwise, readers are referred to [21].

Theorem 1 (Total Order): 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 $v \preceq u$.

Following the approach for representing factorable functions in [21], 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 system of factorable constraints, we therefore call it a *directed acyclic graph* (DAG) for short. In a DAG, every node represents a variable or an elementary operation (e.g. $+$, $\times$, $\div$, log, exp, sqr, sin) used in the composition of constraints and every edge represents the computational flow associated with a coefficient. The ordering of edges is needed for non-commutative operations like the division, and not really necessary for commutative operations. For convenience, a virtual ground node, $\mathbf{G}$, is added to a DAG to be the parent of all the nodes representing the constraints.

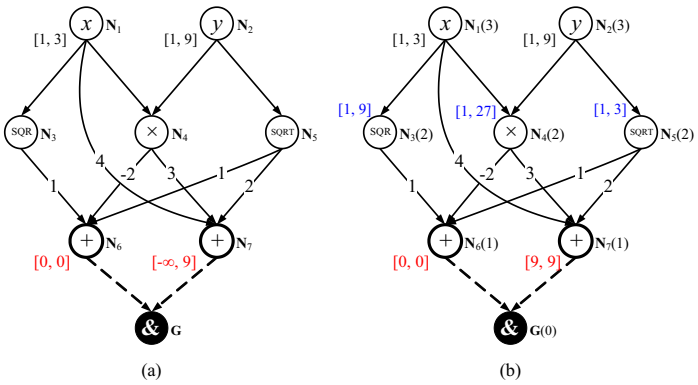

Fig. 1. The DAG representation (a) before and (b) after interval evaluations

We use *multigraphs* instead of simple graphs for the representation because some n-ary operations can take the same input more than once. For example, x^x can be represented by the operation x^y without introducing a unary operation x^x.

Notation 1: In this paper, the real variable representing a node, $\mathbf{N}$, of the DAG representation is denoted by $\vartheta_{\mathbf{N}}$.

Example 1: The DAG representation of the following constraint system is depicted in Figure 1:

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

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 that satisfies the condition of Theorem 1.

C. Fundamental Notations

In the rest of the paper, we use the notation '$\equiv$' for definitions. The power set of a set, A, is denoted by 2^A.

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 notation $E = \langle E \rangle \pm e$ 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 [18] for some rounding techniques in floating-point arithmetic on simple elementary operations.

III. GENERALIZATION OF INCLUSION CONCEPTS

We now generalize the concepts related to *inclusion function* to give a common view of inclusion techniques.

Definition 1 (Inclusion Representation): Given a set $\mathcal{A}$. A couple $\mathcal{I} = (\mathcal{R}, \mu)$, where $\mathcal{R}$ is a set of *representation 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 $\mathcal{I}$.

Definition 2 (Real Representation): 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 representation object $T \in \mathcal{R}$ is a tuple consisting of real constants, and the evaluating function μ can be defined as

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

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.

The domains in D_T can be explicitly given by constant domains such as $[a, b]$, or implicitly given by constraints.

Example 2: It is easy to see that the interval form (2) is equivalent to a real inclusion representation of $\mathbb{R}$; where each representation object $T \in \mathcal{R}$ is a couple of reals (a, b), $V_T = (x)$, f_T is an identity function, and $D_T = [a, b]$. That is, the real representation of the evaluating function is defined as

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

Example 3: The affine form (3) can also be viewed as a real inclusion representation of $\mathbb{R}$; where each representation object is a tuple $T = (x_0, \ldots, x_n, 1, \ldots, n)$,[1] $V_T = (\varepsilon_1, \ldots, \varepsilon_n)$ are the variables of the linear function $f_T(\varepsilon_1, \ldots, \varepsilon_n) = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i$, and $D_T = [-1, 1]^n$. Hence, the real representation of μ can be defined as

$$\mu(T) \equiv \{x_0 + \sum_{i=1}^{n} x_i\varepsilon_i \mid (\varepsilon_1, \ldots, \varepsilon_n) \in [-1, 1]^n\} \tag{8}$$

Example 4: Linear relaxations and polyhedral enclosures can also be viewed as real inclusion representations. Indeed, if they are given by the conjunction of m half-spaces (restricted to a domain $\mathbf{B}$ which is usually a box)

$$H_i \equiv \{(x_1, \ldots, x_n) \mid a_{i0} + \sum_{j=1}^{n} a_{ij}x_j \leq 0\} \ (i = 1, \ldots, m).$$

We can define $T = (a_{10}, \ldots, a_{1n}, \ldots, a_{m0}, \ldots, a_{mn})$, $D_T = \mathbf{B} \cap (\bigcap_{i=1}^{m} H_i)$, and $\mu(T) \equiv \{x \mid x \in D_T\}$, hence

$$\mu(T) = \mathbf{B} \cap (\bigcap_{i=1}^{m} \{(x_1, \ldots, x_n) \mid a_{i0} + \sum_{j=1}^{n} a_{ij}x_j \leq 0\}) \tag{9}$$

Remark 1: For simplicity, we can use the real representation of the evaluating function of an inclusion function to refer to the inclusion representation itself, when not being confused. For example, we will use the affine form $1.5 + 2.5\varepsilon_2$ to refer to the inclusion representation of the interval [-1, 4] instead of using the tuple $T = (1.5, 2.5, 2)$ as in Example 3.

We now generalize the notion of *inclusion function* in the book [4] to accept the notion of inclusion representation.

[1]For simplicity, we keep all coefficients and indices here, but in implementation only non-zero coefficients and their indices should be stored.

Definition 3 (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 for every $S \subseteq X$ and every $T \in \mathcal{R}_X$ we have

$$S \subseteq \mu_X(T) \Rightarrow f(S) \equiv \{f(x) \mid x \in S\} \subseteq \mu_Y(F(T)) \tag{10}$$

In practice, real-valued functions are often extended in a natural way to evaluate the ranges of functions.

Definition 4 (Natural Extension): Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a factorable function recursively composed of a finite set, E, of elementary operations defined on $\mathbb{R}$. Suppose that $\mathcal{I} = (\mathcal{R}, \mu)$ is an inclusion representation of $\mathbb{R}$ such that there exists a set, $E_{\mathcal{R}}$, of elementary operations defined on $\mathcal{R}$, and exists a *bijection* $\eta : E \to E_{\mathcal{R}}$.[2] A function $\mathbf{f} : \mathcal{R}^n \to \mathcal{R}^m$ is called the *natural extension* of f in $\mathcal{I}$ (using elementary operations in $E_{\mathcal{R}}$) if $\mathbf{f}$ is constructed from the composition of f by replacing each real variable (resp. constant) by a variable taking values (resp. constant) in $\mathcal{R}$, and replacing each occurrence of an elementary operation $e \in E$ by the corresponding occurrence of $\eta(e)$. If $\mathbf{f}$ is also an inclusion function of f, we call $\mathbf{f}$ the *natural inclusion function* of f.

Various interval inclusion functions have been described in detail in the book [4]; some among them are *natural, centered, mixed-centered* and *Newton inclusion functions*.

Definition 5 (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 $\forall S \in \mathcal{R}_1 : \mu_1(S) \subseteq \mu_2(c(S))$.

Example 5: Converting the affine form $3 + 2\varepsilon_1 + 1\varepsilon_2$ into interval form, we get [0, 6]. However, converting the interval form [0, 6] into affine form we may get $3 + 3\varepsilon_3$.

Theorem 2 (Composite Inclusion Function): 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 $g \circ f$.

Proof: The proof directly follows Definition 1 and Definition 3, and is hence omitted. ■

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

Proof: Corollary 1 is a straightforward consequence of Theorem 2. The proof is therefore omitted. ■

In implementation, the elementary operations in interval arithmetic and affine arithmetic are constructed to be inclusion functions of their real-valued counterparts. Therefore, as a consequence of Corollary 1, all the factorable operations/functions defined in interval arithmetic (or affine arithmetic) using those operations are also inclusion functions of their real-valued counterparts.

[2]We then call $e \in E$ the real-valued counterpart of $\eta(e)$.

IV. Revised Affine Arithmetic

A. Revised Affine Form

One of the limitations 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 [19], [7], [8], we use a revised affine form similar to (5) but the new term $z_k\varepsilon_k$ is replaced by an 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 as

$$\hat{x} \equiv x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1, 1], \tag{11}$$

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 operation $\hat{z} \equiv \alpha\hat{x} + \beta\hat{y} + \gamma$ is now defined as

$$\hat{z} \equiv (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n} (\alpha x_i + \beta y_i)\varepsilon_i + (|\alpha|e_x + |\beta|e_y)[-1, 1] \tag{12}$$

Note that during computations the lengths of revised affine forms do 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 (12) can be interpreted as follows

$$z_0 = \langle \alpha x_0 + \beta y_0 + \gamma \rangle \pm e_0, \; z_{i=\overline{1,n}} = \langle \alpha x_i + \beta y_i \rangle \pm e_i, \tag{13a}$$

$$e_z = \lceil |\alpha|e_x + |\beta|e_y + \sum_{i=0}^{n} e_i \rceil. \tag{13b}$$

Similarly to the standard affine form (see Example 3), the revised affine form (11) can also be seen as a real inclusion representation of $\mathbb{R}$; where each representation object is a tuple $T = (x_0, \ldots, x_n, 1, \ldots, n, e_x)$, $V_T = (\varepsilon_1, \ldots, \varepsilon_n, \varepsilon_x)$ are the variables of the linear function $f_T(\varepsilon_1, \ldots, \varepsilon_n, \varepsilon_x) = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x\varepsilon_x$, and $D_T = [-1, 1]^{n+1}$. The real representation of the evaluating function can be defined as

$$\mu(T) \equiv \{x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x\varepsilon_x \mid \varepsilon_{i=\overline{1,n}}, \varepsilon_x \in [-1, 1]\} \tag{14}$$

Another limitation 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 needed in 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

$$\hat{x} \equiv \begin{cases} (-\infty, +\infty) & \text{if } e_x = +\infty, \\ (-\infty, x_0] & \text{if } x_\infty = -1, \\ [x_0, +\infty) & \text{if } x_\infty = +1, \\ x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x[-1, 1] & \text{otherwise.} \end{cases} \tag{15}$$

Remark 2: In an operation, if the domain of a variable is unbounded, i.e. in the first three cases of (15), 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 (15).

Notation 2: In this paper, we denote by $\mathbb{A}$ the set of all objects in revised affine form.

B. Unary Operations

We give the following constructive theorem, which is based from a non-constructive theorem named Theorem 2 in [18], 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) If $\forall x \in [a, b] : \alpha \geq f'(x)$, then
   $\forall x \in [a, b] : \alpha x + d_\alpha(b) \leq f(x) \leq \alpha x + d_\alpha(a)$.
2) If f' is continuous and 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 for every $x \in [a, b]$ we have
   $\alpha x + g(\alpha) \leq f(x) \leq \alpha x + \max\{d_\alpha(a), d_\alpha(b)\}$.
3) If f' is continuous and 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 for every $x \in [a, b]$ we have
   $\alpha x + \min\{d_\alpha(a), d_\alpha(b)\} \leq f(x) \leq \alpha x + g(\alpha)$.

Proof: See the proof of Theorem 3 in [22]. ■

To illustrate the usefulness of Theorem 3, in Table I we give the functions f' and g for some elementary functions. In Figure 2, we propose 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: See the proof of Proposition 1 in [22]. ■

Readers are referred to Section 2 of [23] for affine approximations of non-differentiable functions.

TABLE I
FUNCTIONS $f \in \mathcal{C}^1([a, b])$ SATISFYING THE CONDITIONS OF THEOREM 3

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

procedure AApprox(**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. 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 in Theorem 3

C. Multiplication

Similar to the products of two G intervals in [7], [8] (time complexities are $O(n^2)$ and $O(n)$, respectively), 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

$$u = \sum_{i=1}^{n} |x_i|,\ v = \sum_{i=1}^{n} |y_i|, \tag{16a}$$

$$z_0 = x_0 y_0 + \frac{1}{2}\sum_{i=1}^{n} x_i y_i,\ z_i = x_0 y_i + y_0 x_i\ (i = \overline{1,n}), \tag{16b}$$

$$e_z = e_x e_y + e_y(|x_0| + u) + e_x(|y_0| + v) + uv - \frac{1}{2}\sum_{i=1}^{n} |x_i y_i| \tag{16c}$$

This is similar to, but tighter than, the formula for multiplication in [8] when exactly porting into revised affine form. The time complexity of (12) is $O(n)$. In rigorous computing, we use the following computations:

$$u = \lceil \sum_{i=1}^{n} |x_i| \rceil,\ v = \lceil \sum_{i=1}^{n} |y_i| \rceil \tag{17a}$$

$$z_0 = \langle x_0 y_0 + 0.5 \sum_{i=1}^{n} x_i y_i \rangle \pm e_0;\ z_{i=\overline{1,n}} = \langle x_0 y_i + y_0 x_i \rangle \pm e_i \tag{17b}$$

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

Proposition 2: The affine multiplication defined by (16) or by (17) is an inclusion function of the real multiplication.

Proof: See the proof of Proposition 2 in [22]. ■

D. Division

In 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 mentioning that in [8], [24], the authors have proposed better dividing methods.

V. Combining Multiple Inclusion Representations

A. Node Evaluations and Pruning Constraint Systems

The input constraint system is represented by a DAG as described in Section II-B. The computational data stored at each node, $\mathbf{N}$, of the DAG representation consist of a representation object for each real inclusion representation $\mathcal{I} = (\mathcal{R}, \mu)$ of $\mathbb{R}$ and a *constraint range of node* (hereafter called a *node range* for short) which is often an interval.

Notation 3: Let $\mathcal{I} = (\mathcal{R}, \mu)$ be a real inclusion representation of $\mathbb{R}$. We denote by $\tau(\mathbf{N})$ the node range of $\mathbf{N}$, and by $\mathcal{R}(\mathbf{N})$ the representation object of $\mathcal{I}$ that is stored at node $\mathbf{N}$.

Hereafter, we present a concept that allows evaluating node ranges based on child nodes, that is generalized from the *forward evaluation* in [3].

Definition 6 (Node Evaluation, NEV): Let $\mathbf{N}$ be a node of the DAG representation of 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 $\mathbf{N}$, and $\mathcal{I} = (\mathcal{R}, \mu)$ a real inclusion representation. Also let $f_{\mathcal{I}} : \mathcal{R}^k \to \mathcal{R}$ be an inclusion function of f. The following assignment is called the *node evaluation* at node $\mathbf{N}$ in the inclusion representation $\mathcal{I}$ (if $\mathbf{N} \neq \mathbf{G}$):

$$\texttt{NEV}(\mathbf{N}, \mathcal{I}) \equiv \begin{Bmatrix} \mathcal{R}(\mathbf{N}) := \mathcal{R}(\mathbf{N}) \cap \tau(\mathbf{N}) \cap f_{\mathcal{I}}(\{\mathcal{R}(\mathbf{C}_i)\}_{i=1}^k); \\ \tau(\mathbf{N}) := \tau(\mathbf{N}) \cap \mu(\mathcal{R}(\mathbf{N})); \end{Bmatrix}$$

Example 6: Considering Example 1, the nodes of the DAG representation are named $\mathbf{G}$ and $\mathbf{N}_i$ $(i = 1, \ldots, 7)$ as depicted in Figure 1. At the beginning we have (see Figure 1a)

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

The elementary operation corresponding to node $\mathbf{N}_3$ is the square operation, therefore, we have

$$\texttt{NEV}(\mathbf{N}_3, \mathbb{I}) \equiv \begin{Bmatrix} \mathbb{I}(\mathbf{N}_3) := \mathbb{I}(\mathbf{N}_3) \cap \tau(\mathbf{N}_3) \cap (\mathbb{I}(\mathbf{N}_1))^2; \\ \tau(\mathbf{N}_3) := \tau(\mathbf{N}_3) \cap \mathbb{I}(\mathbf{N}_3); \end{Bmatrix}$$

$$\texttt{NEV}(\mathbf{N}_3, \mathbb{A}) \equiv \begin{Bmatrix} \mathbb{A}(\mathbf{N}_3) := \mathbb{A}(\mathbf{N}_3) \cap \tau(\mathbf{N}_3) \cap (\mathbb{A}(\mathbf{N}_1))^2; \\ \tau(\mathbf{N}_3) := \tau(\mathbf{N}_3) \cap \mathbb{A}(\mathbf{N}_3); \end{Bmatrix}$$

After the evaluation $\texttt{NEV}(\mathbf{N}_3, \mathbb{I}) \cap \texttt{NEV}(\mathbf{N}_3, \mathbb{A})$, we have

$\mathbb{I}(\mathbf{N}_3) = [-\infty, +\infty] \cap [-\infty, +\infty] \cap ([1,3])^2 = [1,9]$; $\tau(\mathbf{N}_3) = [1,9]$
$\mathbb{A}(\mathbf{N}_3) = [-\infty, +\infty] \cap [1,9] \cap (2 + \varepsilon_1)^2 = 4.5 + 4\varepsilon_1 + 0.5[-1,1]$
$\tau(\mathbf{N}_3) = [1,9] \cap \mathbb{I}(4.5 + 4\varepsilon_1 + 0.5[-1,1]) = [1,9]$

Similarly, after performing node evaluations at the other nodes we have $\mathbb{I}(\mathbf{N}_i) = \tau(\mathbf{N}_i)$ for every $i = 1, \ldots, 7$ and

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

In order to present the concept of pruning constraint systems concisely, we rely on the following notion.

Definition 7 (Inclusion Constraint System, ICS): Let $(\mathcal{R}, \mu)$ be a real inclusion representation of $\mathbb{R}$ defined by (6), $\mathbf{N}$ a node of the DAG representation. The *inclusion constraint system* induced by a representation object $T \equiv \mathcal{R}(\mathbf{N})$ and a *constraint range* $D \subseteq \mathbb{R}$ is defined as

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

where the set of variables of the inclusion constraint system consists of the variable $\vartheta_{\mathbf{N}}$, the variables in V_T, and the variables used to describe D_T.

Example 7: We give some inclusion constraint systems:

- for the interval form (7):

$$\texttt{ICS}(T,[c,d]) \equiv \{\vartheta_{\mathbf{N}} \in [c,d] \cap [a,b]\},$$

where the set of variables is $\{\vartheta_{\mathbf{N}}\}$. This system is conjunctive and has the form of bound constraints.

- for the revised affine form (14):

$$\texttt{ICS}(T,[c,d]) \equiv \{x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x\varepsilon_x = \vartheta_{\mathbf{N}}; \vartheta_{\mathbf{N}} \in [c,d]; \\ (\varepsilon_1,\ldots,\varepsilon_n,\varepsilon_x) \in [-1,1]^{n+1}\},$$

where the set of variables is $\{\varepsilon_1,\ldots,\varepsilon_n,\varepsilon_x,\vartheta_{\mathbf{N}}\}$. This system is conjunctive and linear.

- for the linear relaxations/polyhedral enclosures in (9):

$$\texttt{ICS}(T,[c,d]) \equiv \{a_{i0} + \sum_{j=1}^{n} a_{ij}x_j \leq 0 \ (i=1,\ldots,m); \\ \vartheta_{\mathbf{N}} \in [c,d];\ (x_1,\ldots,x_n) \in \mathbf{B}\},$$

where the set of variables is $\{x_1,\ldots,x_n,\vartheta_{\mathbf{N}}\}$. This system is conjunctive and linear.

We now present the construction of constraint systems for pruning node ranges based on representation objects.

Definition 8 (Pruning Constraint System, PCS*):* Let $\mathbf{N}$ be a node of the DAG representation, $\{\mathbf{C}_i\}_{i=1}^{k}$ the children of $\mathbf{N}$, $f : \mathbb{R}^k \to \mathbb{R}$ the elementary operation represented by $\mathbf{N}$, and $\mathcal{S}$ a finite set of real inclusion representations. The following constraint system is called the *pruning constraint system* induced by the inclusion representations of $\mathcal{S}$ at $\mathbf{N}$:

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

where $\texttt{PCSub}(\mathbf{N},\mathcal{R},\mu)$ is a *pruning inclusion subsystem*:

$$\texttt{PCSub}(\mathbf{N},\mathcal{R},\mu) = \texttt{ICS}(\mathcal{R}(\mathbf{N}),\tau(\mathbf{N})) \wedge \bigwedge_{i=1}^{k} \texttt{ICS}(\mathcal{R}(\mathbf{C}_i),\tau(\mathbf{C}_i)).$$

Notation 4: In the rest, we will abuse the notations $\mathbb{I}$ and $\mathbb{A}$ to denote the real inclusion representations, $(\mathbb{I},\mu_{\mathbb{I}})$ and $(\mathbb{A},\mu_{\mathbb{A}})$, respectively defined on interval arithmetic and revised affine arithmetic; where the function $\mu_{\mathbb{I}}$ is defined by (7) and the function $\mu_{\mathbb{A}}$ is defined by (14).

Example 8: Considering Example 1, we have, for instance, the following inclusion constraint systems:

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_4),\tau(\mathbf{N}_4)) \equiv \begin{cases} 10 + 5\varepsilon_1 + 8\varepsilon_2 + 4\varepsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4}; \\ \vartheta_{\mathbf{N}_4} \in [1,27]; (\varepsilon_1,\varepsilon_2,\varepsilon_{\mathbf{N}_4}) \in [-1,1]^3 \end{cases}$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_5),\tau(\mathbf{N}_5)) \equiv \begin{cases} 2.125 + \varepsilon_2 + 0.125\varepsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5}; \\ \vartheta_{\mathbf{N}_5} \in [1,3]; (\varepsilon_2,\varepsilon_{\mathbf{N}_5}) \in [-1,1]^2 \end{cases}$$

$$\texttt{ICS}(\mathbb{A}(\mathbf{N}_7),\tau(\mathbf{N}_7)) \equiv \begin{cases} 42.25 + 19\varepsilon_1 + 26\varepsilon_2 + 12.25\varepsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7}; \\ \vartheta_{\mathbf{N}_7} \in [9,9]; (\varepsilon_1,\varepsilon_2,\varepsilon_{\mathbf{N}_7}) \in [-1,1]^3 \end{cases}$$

and the following pruning constraint systems

$$\texttt{PCS}(\mathbf{N}_7,\{\mathbb{I}\}) \equiv \begin{cases} 4\vartheta_{\mathbf{N}_1} + 3\vartheta_{\mathbf{N}_4} + 2\vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_7} \\ \vartheta_{\mathbf{N}_1} \in [1,3]; \vartheta_{\mathbf{N}_4} \in [1,27]; \\ \vartheta_{\mathbf{N}_5} \in [1,3]; \vartheta_{\mathbf{N}_7} \in [9,9] \end{cases}$$

$$\texttt{PCS}(\mathbf{N}_7,\{\mathbb{A}\}) \equiv \begin{cases} 4\vartheta_{\mathbf{N}_1} + 3\vartheta_{\mathbf{N}_4} + 2\vartheta_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_7} \\ 2 + \varepsilon_1 = \vartheta_{\mathbf{N}_1} \\ 10 + 5\varepsilon_1 + 8\varepsilon_2 + 4\varepsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4} \\ 2.125 + \varepsilon_2 + 0.125\varepsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5} \\ 42.25 + 19\varepsilon_1 + 26\varepsilon_2 + 12.25\varepsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7} \\ \vartheta_{\mathbf{N}_1} \in [1,3]; \vartheta_{\mathbf{N}_4} \in [1,27]; \\ \vartheta_{\mathbf{N}_5} \in [1,3]; \vartheta_{\mathbf{N}_7} \in [9,9] \\ (\varepsilon_1,\varepsilon_2,\varepsilon_{\mathbf{N}_4},\varepsilon_{\mathbf{N}_5},\varepsilon_{\mathbf{N}_7}) \in [-1,1]^5 \end{cases}$$

B. A Scheme for Combining Inclusion Representations

In this section, we describe a generic combination scheme that combines the strength of different real inclusion representations for constraint propagation. In this scheme each input constraint system, say an NCSP, is represented by a DAG as described in Section II-B. The computational data stored at each node are the representation objects as described in Section V-A. In principle, the scheme uses the node evaluations and pruning constraint systems, which are defined in Section V-A, and uses relevant pruning techniques to reduce the node ranges, hence, reduce the variable domains.

Definition 9 (Pruning Technique): A *pruning technique* for a real constraint system is a technique for reducing some domains of the variables of the constraint system.

Let $\mathcal{G}$ be a DAG which represents the input constraint system. The proposed scheme, called CIRD, uses two waiting lists. The first waiting list, denoted by $\mathcal{L}_e$, stores the nodes waiting for evaluation. The second waiting list, denoted by $\mathcal{L}_p$, stores the nodes waiting for node pruning. Note that each node can appear once at a time in one waiting list, but may appear in both waiting lists. The set of real inclusion representations for use in the scheme is denoted by $\mathcal{E}$. We suppose that each real inclusion representation in $\mathcal{E}$ provides elementary operations which are inclusion functions of their real-valued counterparts. In Figure 3, we present the main steps of the CIRD scheme with inline detailed descriptions.

Proposition 3: We define a function $F : \mathbb{I}^n \times 2^{\mathbb{R}^n} \to \mathbb{I}^n$ to represent the CIRD algorithm. This function takes as input the variable domains (in the form of an interval box $\mathbf{B}$) and the exact solution set, S, of the input problem. The function F returns an interval box, denoted by $F(\mathbf{B},S)$, that represents the variable domains of the output of the CIRD algorithm. The CIRD algorithm terminates at a finite number of iterations and the following properties hold:

(i) $F(\mathbf{B},S) \subseteq \mathbf{B}$ (Contractiveness)

(ii) $F(\mathbf{B},S) \supseteq \mathbf{B} \cap S$ (Correctness)

Proof: The proof is trivial due to the finite nature of floating-point numbers and the fact that the node ranges are never inflated during the computations. ■

VI. Specific Combination Strategies as Instances

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 strategies for constraint propagation.

1) **Initialization Phase.**
   a) **Initial Node Evaluation.** Select an algorithm for visiting DAGs in an order 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 $\mathcal{I} \in \mathcal{E}$. Merging the assignments of multiple $\texttt{NEV}(\mathbf{N}, \mathcal{I})$ into a single process to avoid repeating the same computations is encouraged.
   b) **Initialize Waiting Lists.** Set $\mathcal{L}_e := \emptyset$, $\mathcal{L}_p := \{$the list of all nodes representing the active constraints associated with all real inclusion representations of $\mathcal{E}\}$.
2) **Propagation Phase.** Repeat this step until both $\mathcal{L}_e$ and $\mathcal{L}_p$ become empty.
   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: (combining several inclusion representations for better evaluation by using inclusion converters is also an option)
      i) Perform the node evaluation $\texttt{NEV}(\mathbf{N}, \mathcal{I})$. If this returns an empty set, the algorithm terminates 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) $\texttt{PCS}(\mathbf{N}, \mathcal{U})$. Subsequently, 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 terminates with an infeasible status.
      iii) Let $\mathcal{K}$ be the set of all the nodes whose evaluating functions in the 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 the 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 terminates 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 $\mathcal{L}_p$.

Fig. 3. `CIRD` – a generic scheme for *Combining Inclusion Representations on DAGs*

procedure `RecursiveNodeEval`(**in** : $\mathbf{N}$)
  if $\mathbf{N}$ is a leaf **or** $\mathbf{N}$ has been visited **then return**;
  for each $\mathbf{C} \in children(\mathbf{N})$ **do** $\texttt{RecursiveNodeEval}(\mathbf{C})$;
  for each $\mathcal{I} \in \mathcal{E}$ **do** $\texttt{NEV}(\mathbf{N}, \mathcal{I})$;
  Mark $\mathbf{N}$ as visited;
  if empty is detected **then exit**(*infeasible*);
end

Fig. 4. The pseudo code for recursive node evaluation

procedure `NodeLevel`(**in** : $\mathbf{N}$; **in/out** : V_{lvl})
  for each $\mathbf{C} \in children(\mathbf{N})$ **do**
    $V_{lvl}[\mathbf{C}] := \max\{V_{lvl}[\mathbf{C}], V_{lvl}[\mathbf{N}] + 1\}$;
    $\texttt{NodeLevel}(\mathbf{C}, V_{lvl})$;
  end-for
end

Fig. 5. The procedure to assign a node level to each node in the DAG representation.

A. Step 1: Initial Node Evaluation and Waiting Lists

In our implementation, we use a recursive evaluation procedure given in Figure 4 for the visit at Step 1a. If this procedure exit with an infeasible status, the main algorithm invoking it terminates with an infeasible status.

Example 9: We continue to consider Example 1. After the initial node evaluation using interval arithmetic and revised affine arithmetic, we have the node ranges given in Example 6. After perform Step 1b with $\mathcal{S} = \{\mathbb{I}, \mathbb{A}\}$ we have $\mathcal{L}_e = \emptyset$ and $\mathcal{L}_p = \{(\mathbf{N}_6; \mathbb{I}, \mathbb{A}), (\mathbf{N}_7; \mathbb{I}, \mathbb{A})\}$.

B. 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 node has a level smaller than that of their descendants. Hence, an ordering in Theorem 1 can be obtained easily by sorting in node levels.

Figure 5 gives a simple procedure to compute a vector V_{lvl} of node levels if this procedure is invoked at all the nodes representing the active constraints. Figure 1 illustrates the node levels for the constraint system given in Example 1. The node levels are given in brackets next to the node names.

The list $\mathcal{L}_p$ is sorted in the ascending order of node levels. It is to maintain that each node being taken into pruning processes before its descendants. Similarly, the list $\mathcal{L}_e$ is sorted in the descending order of node levels to maintain that each node being evaluated before its ancestors.

There are two simple strategies to get the next node from the two waiting lists $\{\mathcal{L}_e, \mathcal{L}_p\}$ as follows: (i) get the next node from $\mathcal{L}_p$ whenever $\mathcal{L}_p$ is not empty, this is called the "pruning first" strategy; and (ii) 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 "pruning first" strategy. More complicated strategies for choosing the next node can be used as alternatives, for example, based on the pruning efficiency of nodes.

C. Step 2b: Node Evaluation

For the node evaluation at each node $\mathbf{N}$, we can perform $\texttt{NEV}(\mathbf{N}, \mathbb{A})$ and $\texttt{NEV}(\mathbf{N}, \mathbb{I})$ in any order, if $\mathbf{N}$ 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 $d_w > 0$ [25]. More complicated criteria can also be used as alternatives.

D. 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 $\texttt{PCS}(\mathbf{N}, \{\mathbb{A}\})$:

$$\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\}) \equiv \left\{\begin{array}{l} f(\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k}) = \vartheta_{\mathbf{N}}; \\ \vartheta_{\mathbf{N}} \in \tau(\mathbf{N}); \\ \bigwedge_{i=1}^{k} (\vartheta_{\mathbf{C}_i} \in \tau(\mathbf{C}_i)); \end{array}\right\} \text{ if } \mathbf{N} \neq \mathbf{G};$$

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

where

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

Note that we have in general

$$\texttt{PCS}(\mathbf{M}, \{\mathbb{A}\}) \equiv \texttt{PCS}(\mathbf{M}, \{\mathbb{I}\}) \wedge \texttt{PCS}_L(\mathbf{M}, \{\mathbb{A}\}). \quad (18)$$

Example 10: We now consider Example 1. Some examples of pruning constraint systems are also given in Example 8.

$$\texttt{PCS}_L(\mathbf{N}_7, \{\mathbb{A}\}) \equiv \begin{cases} 2 + \varepsilon_1 = \vartheta_{\mathbf{N}_1} \\ 10 + 5\varepsilon_1 + 8\varepsilon_2 + 4\varepsilon_{\mathbf{N}_4} = \vartheta_{\mathbf{N}_4} \\ 2.125 + \varepsilon_2 + 0.125\varepsilon_{\mathbf{N}_5} = \vartheta_{\mathbf{N}_5} \\ 42.25 + 19\varepsilon_1 + 26\varepsilon_2 + 12.25\varepsilon_{\mathbf{N}_7} = \vartheta_{\mathbf{N}_7} \\ \vartheta_{\mathbf{N}_1} \in [1,3]; \vartheta_{\mathbf{N}_4} \in [1,27]; \vartheta_{\mathbf{N}_5} \in [1,3]; \\ \vartheta_{\mathbf{N}_7} \in [9,9]; (\varepsilon_1, \varepsilon_2, \varepsilon_{\mathbf{N}_4}, \varepsilon_{\mathbf{N}_5}, \varepsilon_{\mathbf{N}_7}) \in [-1,1]^5 \end{cases}$$

The node ranges are pruned by using a combination of backward propagation and affine pruning techniques as follows.

1) Backward Propagation: If $\mathbf{N}$ is not the ground, the domains of the variables of the constraint system $\texttt{PCS}(\mathbf{N}, \{\mathbb{I}\})$ can be pruned by a pruning technique which is called *backward propagation* in [3], [21]. In brief, let f be the elementary operation represented by a node $\mathbf{N}$, we then have the relation $\vartheta_{\mathbf{N}} = f(\{\vartheta_{\mathbf{C}_i}\}_{i=1}^{k})$. For each i in $\{1, \ldots, k\}$, the backward propagation computes a cheap evaluation of the i-th projection of the relation $\vartheta_{\mathbf{N}} = f(\{\vartheta_{\mathbf{C}_i}\}_{i=1}^{k})$ onto $\vartheta_{\mathbf{C}_i}$. In case there exist a function $g_i : \mathbb{R}^k \to \mathbb{R}$ such that we can write $\vartheta_{\mathbf{C}_i} = g_i(\vartheta_{\mathbf{N}}, \{\vartheta_{\mathbf{C}_j}\}_{j=1;j\neq i}^{k})$. Let G_i be an inclusion function of g_i in $\mathbb{I}$. In this case, the backward propagation at $\mathbf{N}$ is defined as $\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap G_i(\mathbb{I}(\mathbf{N}), \{\mathbb{I}(\mathbf{C}_j)\}_{j=1;j\neq i}^{k})$ $(i = \overline{1,k})$. A deeper discussion on the other cases can be found in [25].

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.

2) Affine Pruning: Each variable of the input constraint system is associated with one noise symbol ε_i $(i = \overline{1,n})$ in $\mathbb{A}$. 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 [26].

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 $\{\vartheta_{\mathbf{C}_i}\}_{i=1}^{k}$ 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 $\{\vartheta_{\mathbf{C}_i}\}_{i=1}^{k}$, 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 $\{\vartheta_{\mathbf{C}_i}\}_{i=1}^{k}$ 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

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

Remark 3: The cost of linear programming is high, 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 $(\varepsilon_1, \ldots, \varepsilon_n)$. Moreover, this type of pruning should only be used for nodes at low levels.

VII. Experiments

A. Comparisons with Linear Relaxation based Techniques

We first compare the proposed technique with a recent mathematical solving technique, called A2, in [8] which was specially designed to solve nonlinear equation systems. The A2 algorithm converts an equation system into *separable form*, and 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, b is a real vector, and $\mathbf{x}$ and $\mathbf{y}$ are interval vectors. This technique has to assume a posterior-condition that A is *invertible* in order to use the reduction rule $\mathbf{x}' := \mathbf{x} \cap (A^{-1}By - A^{-1}b)$. No rigorous rounding technique is found in [8]. We take the first problem that was used for illustrating the power of the A2 algorithm in [8] for the comparison:

$$\begin{cases} ((4x_3 + 3x_6)x_3 + 2x_5)x_3 + x_4 = 0, \\ ((4x_2 + 3x_6)x_2 + 2x_5)x_2 + x_4 = 0, \\ ((4x_1 + 3x_6)x_1 + 2x_5)x_1 + x_4 = 0,\ x_4 + x_5 + x_6 + 1 = 0 \\ (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_2 + (((x_3 + x_6)x_3 + x_5)x_3 + x_4)x_3 = 0, \\ (((x_1 + x_6)x_1 + x_5)x_1 + x_4)x_1 + (((x_2 + x_6)x_2 + x_5)x_2 + x_4)x_3 = 0, \\ x_1 \in [0.0333, 0.2173],\ x_2 \in [0.4000, 0.6000], \\ x_3 \in [0.7826, 0.9666], x_4 \in [-0.3071, -0.1071], \\ x_5 \in [1.1071, 1.3071],\ x_6 \in [-2.1000, -1.9000] \end{cases} \quad (20)$$

The system (20) is known to be hard for interval techniques and has a unique solution. To solve it on a 1.7 GHz Pentium PC at the resolution 10^{-5} using a bisection search; A2 has to perform 917 iterations in 3.46 seconds to reduce the problem

TABLE II

A PRELIMINARY COMPARISON BETWEEN Quad AND CIRD[ai]

Propagator ▶	Quad				CIRD[ai]				Time ratio Quad / CIRD[ai]
Problem ▼	$\#S$	$\#B$	Time (sec.)	CPU (GHz)	$\#S$	$\#B$	Time (sec.)	CPU (GHz)	
Gough-Steward (9)	24	4	183.0	1.0	912	4	2.7	1.7	39.9
Yama196 ($n = 30$)	108	16	31.4	2.66	25	2	3.8	1.7	12.9
Yama196 ($n = 60$)	n/a	n/a	n/a	0.8	18	2	21.0	1.7	n/a
Yama196 ($n = 100$)	n/a	n/a	n/a	0.8	20	2	85.8	1.7	n/a
Yama196 ($n = 200$)	n/a	n/a	n/a	0.8	19	2	560.2	1.7	n/a
Yama196 ($n = 300$)	n/a	n/a	n/a	0.8	20	2	1878.1	1.7	n/a

to 5 boxes (see [8]); while an instance of the CIRD scheme, called CIRD[ai],[3] performs 54 iterations in only 0.118 seconds to reduce the problem to 3 boxes. Hence, CIRD[ai] is about 29.3 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 [5], which was specifically designed to process quadratic constraints, and an extension of Quad in [6]. Again, we take as example two problems, called *Gough-Steward* and *Yama196*, which were used to illustrate the power of Quad in [5], [6], respectively. *Gough-Steward* is a non-sparse quadratic equation system of 9 variables in Robotics, which has four solutions [5]. *Yama196* is a series of high-dimensional sparse problems of n variables and n equations of the form $\{(n+1)^2 x_{i-1} - 2(n+1)^2 x_i + (n+1)^2 x_{i+1} + e^{x_i} = 0,\ x_i \in [-10, 10] \mid i = 1, \ldots, n\}$, where $x_0 = x_{n+1} = 0$. Similarly to [6], we use the resolution 10^{-8} for these problems. Table II presents a preliminary comparison between CIRD[ai] and Quad.

Note 1: The results of Quad in Table II are copied from [5], [6], except that the ones in the cells filled with "n/a" are not yet available due to our limited access to the implemenation of Quad. In Table II, $\#S$ denotes the number of splittings and $\#B$ denotes the number of boxes in the output.

B. Comparisons with Interval Propagation Techniques

We have carried out experiments on an implementation of the CIRD[ai] algorithm (newer than the one in [22]) and two other well-known state-of-the-art interval constraint processing techniques. The first one is an implementation of Box Consistency [27], [28] in a well-known commercial product named ILOG Solver (v6.0, 11/2003), hereafter denoted by BOX. The second one is called HC4 (Revised Hull Consistency) from [3]. The experiments are carried out on 33 problems which are *unbiasedly* chosen and divided into five test cases for analyzing the test results:[4]

- The test case T_1 consists of 8 easy problems with isolated solutions that are solvable by the search using the three propagators in short time.
- The test case T_2 consists of 4 average problems with isolated solutions that are solvable by the search using CIRD[ai] and BOX, and that cause the search using HC4 being out of time without reaching 10^6 splittings.
- The test case T_3 consists of 8 hard problems with isolated solutions that cause the search using HC4 being out of time without reaching 10^6 splittings; and that cause the search using BOX either being out of time or being stopped due to running more than 10^6 splittings. The search using CIRD[ai] accomplishes the solving for six of eight problems in this test case, and runs more than 10^6 splittings for the other two problems.
- The test case T_4 consists of 7 easy problems with a continuum of solutions that are solvable at the predefined resolution 10^{-2} in short time.
- The test case T_5 consists of 6 hard problems with a continuum of solutions that are solvable at the predefined resolution 10^{-1} in short time.

The timeout value is set to **10 hours** for all the test cases. *The timeout values will be used as the running time for the techniques which are out of time 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 the bisection search. For the last two test cases, the search to be used is a search technique, called UCA6, for inequalities (see [9], [10]). The comparison of the interval constraint propagation techniques is based on the measures of

- *The running time:* The relative ratio of the running time of each propagator to that of CIRD[ai] is called the *relative time ratio*.
- *The number of boxes:* The relative ratio of the number of boxes in the output of each propagator to that of CIRD[ai] is called the *relative cluster ratio*.
- *The number of splittings:* The number of splittings in search needed to solve the problems. The relative ratio of the number of splittings used by each propagator to that of CIRD[ai] is called the *relative iteration ratio*.
- *The volume of boxes (only for T_1, T_2, T_3):* We consider the reduction per dimension $\sqrt[d]{V/D}$; where d is the dimension, V is the total volume of the output boxes, D is the volume of the initial domains. The relative ratio of the reduction gained by each propagator to that of CIRD[ai] is called the *relative reduction ratio*.
- *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 III and Table IV.

Note 2: In general, the lower the relative ratio is, the better the performance/quality is; and the higher the inner volume ratio is, the better the quality is. In the section (a) of Table III, the average of the relative time ratios is taken over all the problems in the test cases T_1, T_2, T_3; and 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. In the section (b) of Table III, the averages of the relative ratios are taken over all the problems in the test cases T_4, T_5.

[3] In this paper, we use a new implementation of CIRD[ai], which is an improvement of the old version used in [22].

[4] We have collected a set of problems from diverse sources including related papers and the Internet.

TABLE III

THE COMPARISON OF THE THREE CONSTRAINT PROPAGATION TECHNIQUES IN SOLVING NCSPS (USING THE NEW VERSION OF CIRD[ai])

Prop. ▼	(a) Isolated Solutions				(b) Continuum of Solutions			
	Relative time ratio	Relative reduction ratio	Relative cluster ratio	Relative iteration ratio	Relative time ratio	Inner volume ratio	Relative cluster ratio	Relative iteration ratio
CIRD[ai]	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.945**	**1.000**	**1.000**
BOX	1429.660	5.323	30.206	4.263	3.414	0.944	1.102	1.056
HC4	17283.614	7.722	105.825	5.515	60.101	0.941	1.168	1.118

TABLE IV

THE AVERAGES OF THE RELATIVE TIME RATIOS FOR EACH TEST CASE

Prop. ▼	(a) Isolated Solutions			(b) Contnm. of Solutions	
	Case T_1	Case T_2	Case T_3	Case T_4	Case T_5
CIRD[ai]	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
BOX	8.33	6097.45	517.10	2.33	4.68
HC4	54.47	83009.81	1649.66	31.42	93.56

Clearly, CIRD[ai] is superior than BOX and HC4 in performance and quality for the problems with isolated solutions in the unbiasedly chosen benchmarks. CIRD[ai] still outperforms the others for the problems with continuum of solutions while being a little better than the others in quality of the output.

VIII. CONCLUSION

In this paper, we propose a novel generic scheme, CIRD, for constraint propagation using different inclusion representations on DAG. The scheme is able to incorporate most of known inclusion representations, including interval arithmetic, affine arithmetic, polyhedral/quadratic enclosures and their generalizations. Modifications and improvements of the rigorous computations of affine arithmetic are also proposed. As a result, we give several new combination strategies for constraint propagation based on interval arithmetic, affine arithmetic, interval constraint propagation and *safe* linear programming. We then show by experiments that an implementation, CIRD[ai], outperforms recent techniques by 1–4 orders of magnitude or more in speed, while still being better in quality measures. A potential direction for future is to integrate the *quadratic form* [19] or linear relaxations [29] into the CIRD scheme.

ACKNOWLEDGEMENTS

This research is funded through the COCONUT project (IST-2000-26063). We would like to thank ILOG for the licenses of ILOG Solver/CPLEX, and thank the COCONUT team of the University of Nantes for the HC4 code. We specially thank Dr. Hermann Schichl and Prof. Arnold Neumaier for encouraging us to use the DAG representation for constraint propagation [21] during the COCONUT project.

REFERENCES

[1] F. Benhamou and W. J. Older, "Applying Interval Arithmetic to Real, Integer and Boolean Constraints," *Journal of Logic Programming*, pp. 32–81, 1997.

[2] P. Van Hentenryck, "Numerica: A Modeling Language for Global Optimization," in *Proceedings of IJCAI'97*, 1997.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget, "Revising Hull and Box Consistency," in *Proceedings of the International Conference on Logic Programming (ICLP'99)*, USA, 1999, pp. 230–244.

[4] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*, 1st ed. Springer, 2001.

[5] Y. Lebbah, M. Rueher, and C. Michel, "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)*, 2003, pp. 109–123.

[6] Y. Lebbah, C. Michel, and M. Rueher, "Global Filtering Algorithms Based on Linear Relaxations ," in *Notes of the 2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS 2003)*, Switzerland, November 2003.

[7] L. V. Kolev, "Automatic Computation of a Linear Interval Enclosure," *Realiable Computing*, vol. 7, pp. 17–18, 2001.

[8] ——, "An Improved Interval Linearization for Solving Non-Linear Problems," in *10th Int'l Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN2002)*, September 2002.

[9] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings, "Search Techniques for Non-linear CSPs with Inequalities," in *Proceedings of the 14th Canadian Conference on Artificial Intelligence*, 2001.

[10] X.-H. Vu, D. Sam-Haroud, and M.-C. Silaghi, "Numerical Constraint Satisfaction Problems with Non-isolated Solutions," in *Global Optimization and Constraint Satisfaction*, vol. LNCS 2861. Springer-Verlag, October 2003, pp. 194–210.

[11] M. Warmus, "Caculus of Approximations," *Bulletin de l'Académie Polonaise des Sciences*, vol. IV(5), pp. 253–259, 1956.

[12] T. Sunaga, "Theory of an Interval Algebra and its Applications to Numerical Analysis," *RAAG Memoirs*, vol. 2, pp. 29–46, 1958.

[13] R. E. Moore, "Automatic Error Analysis in Digital Computation," Missiles and Space Division, Lockheed Aircraft Corporation, Sunnyvale, California, USA, Tech. Rep. LMSD-84821, 1959.

[14] G. Alefeld and J. Herzberger, *Introduction to Interval Computations*. New York, NY: Academic Press, 1983.

[15] A. Neumaier, *Interval Methods for Systems of Equations*. Cambridge: Cambridge Univ. Press, 1990.

[16] T. J. Hickey, Q. Ju, and M. H. Van Emden, "Interval Arithmetic: from Principles to Implementation," *Journal of the ACM (JACM)*, vol. 48(5), pp. 1038–1068, 2001.

[17] J. L. D. Comba and J. Stolfi, "Affine Arithmetic and its Applications to Computer Graphics," in *Proceedings of SIBGRAPI'93*, Brazil, 1993.

[18] J. Stolfi and L. H. de Figueiredo, "Self-Validated Numerical Methods and Applications," in *Monograph for 21st Brazilian Mathematics Colloquium (IMPA)*, Brazil, July 1997.

[19] F. Messine, "Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 992–1015, November 2002.

[20] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang, "Comparison of Interval Methods for Plotting Algebraic Curves," *Computer Aided Geometric Design*, vol. 19(7), pp. 553–587, 2002.

[21] H. Schichl and A. Neumaier, "Interval Analysis on Directed Acyclic Graphs for Global Optimization," 2004, preprint - University of Vienna, Autria.

[22] X.-H. Vu, D. Sam-Haroud, and B. Faltings, "A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation," Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland, Tech. Rep. IC/2004/39, April 2004.

[23] L. V. Kolev, "A New Method for Global Solution of Systems of Non-Linear Equations," *Realiable Computing*, vol. 4, pp. 125–146, 1998.

[24] S. Miyajima, T. Miyata, and M. Kashiwagi, "A New Dividing Method in Affine Arithmetic," *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A(9), pp. 2192–2196, September 2003.

[25] X.-H. Vu, H. Schichl, and D. Sam-Haroud, "Using Directed Acyclic Graphs to Coordinate Propagation and Search for Numerical Constraint Satisfaction Problems," in *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. Florida, USA: IEEE Computer Society Press, November 2004.

[26] A. Neumaier and O. Shcherbina, "Safe Bounds in Linear and Mixed-Integer Programming," *Mathematical Programming A*, vol. 99, pp. 283–296, 2004.

[27] F. Benhamou, D. McAllester, and P. Van Hentenryck, "CLP(Intervals) Revisited," in *Proceedings of the International Logic Programming Symposium*, 1994, pp. 109–123.

[28] P. Van Hentenryck, D. McAllester, and D. Kapur, "Solving Polynomial Systems Using a Branch and Prune Approach," *SIAM Journal of Numerical Analysis*, vol. 34(2), 1997.

[29] S. Hongthong and R. B. Kearfott, "Rigorous Linear Overestimators and Underestimators," *Mathematical Programming B*, 2004, submitted.