Using Directed Acyclic Graphs to Coordinate Propagation and Search for Numerical Constraint Satisfaction Problems

Xuan-Ha Vu[1], Hermann Schichl[2], and Djamila Sam-Haroud[1]

[1] Artificial Intelligence Laboratory,
Swiss Federal Institute of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
{xuan-ha.vu, jamila.sam}@epfl.ch
http://liawww.epfl.ch
[2] Faculty of Mathematics,
University of Vienna,
Nordbergstr. 15, A-1090 Wien, Austria
hermann.schichl@esi.ac.at
http://www.mat.univie.ac.at/~herman

Technical Report No. IC/2004/48 (May 31, 2004)

Abstract. The pioneering paper H. Schichl and A. Neumaier [1] has founded the fundamentals of interval analysis on DAGs for global optimization, including a fundamental of constraint propagation. In this paper, we extend the constraint propagation technique for solving numerical constraint satisfaction problems. In particular, we propose an advanced constraint propagation technique, which makes the constraint propagation practical and efficient, and a method to coordinate constraint propagation and exhaustive search, which uses a single DAG for each problem. The experiments carried out on various problems show that the new approach outperforms previously available propagation techniques by an order of magnitude or more in speed, while being roughly the same quality w.r.t. enclosure properties.

1 Introduction

Many real-world problems require solving numerical constraint satisfaction problems (NCSPs). An NCSP is a triplet $(\mathcal{V}, \mathcal{C}, \mathcal{D})$ which consists of a finite set $\mathcal{V}$ of variables taking their values in domains $\mathcal{D}$ over the reals and subject to a finite set $\mathcal{C}$ of *numerical* constraints. A tuple of values assigned to the variables such that all the constraints are satisfied is called a solution. The set of all the solutions is called the solution set. In practice, numerical constraints are often equalities or inequalities expressed in *factorable* form, that is, they can be represented by elementary functions such as $+$, $-$, $\times$, $\div$, log, exp, sin, cos,... 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 size n and m respectively.

Many solution techniques have been proposed in *Constraint Programming* and *Mathematical Programming* to solve NCSPs. To achieve full rigor when dealing with floating-point numbers, most solution techniques have been based on *interval arithmetic* or its variants. In the last ten years, there have been elaborate uses of interval arithmetic to devise the notions of *inclusion test* and *contractor* as described in the book Jaulin *et al.* [2]. An inclusion test is to check the inclusion of variable domains in the solution set. A contractor, possible variants are *narrowing operator*s [3, 4] and *contracting operator*s [5–7], is a method to narrow the variable domains such that no solution is lost. Various basic inclusion tests and contractors have been described in [2]. Recently, there has been a new approach, called *interval constraint propagation*, which associates *constraint propagation/local consistency* techniques in artificial intelligence with interval analytic methods to devise advanced contractors, e.g., the so-called *forward-backward contractor* [4, 2]. A representation of the solutions can be often computed by interleaving inclusion tests or contractors with *exhaustive search*; the solution techniques often use *bisection* search to solve the problems exhaustively. However, advanced search techniques (see Silaghi *et al.* [6] and Vu *et al.* [7]) have also been proposed to improve the search performance for problems with a continuum of solutions (e.g., inequalities), while maintaining the same performance for problems with isolated solutions (e.g., equalities).

Most recently, a fundamental framework for interval analysis on DAGs has been proposed by Schichl & Neumaier [1], which includes an extension of forward-backward propagation for working on DAGs. In order to exploit the framework and make it useful for more applications, in this paper we extend the DAG-based constraint propagation technique for solving NCSPs. In Section 3, we describe the DAG representation of problems and new extensions of the forward evaluation and the backward propagation. In Section 4, we propose an advanced constraint propagation technique, which makes the above framework for constraint propagation efficient and practical, and a method to coordinate constraint propagation and exhaustive search using a single DAG for each problem. Finally, as one can see in Section 5, the experiments carried out on various problems show that the new approach outperforms previously available propagation techniques by an order of magnitude or more in speed, while being roughly the same quality w.r.t. enclosure properties.

2 Background

2.1 Interval Arithmetic

Interval arithmetic is an extension of real arithmetic defined on the set of real intervals, rather than the set of real numbers. According to Kearfott [8], a form of interval arithmetic perhaps first appeared in 1924 in Burkill [9]. Modern development of interval arithmetic began with R. E. Moore's dissertation [10].

Fundamentally, if $\mathbf{x}$ and $\mathbf{y}$ are two real intervals, then the four elementary operations for *idealized interval arithmetic* obey the rule: $\mathbf{x} \diamond \mathbf{y} = \{x \diamond y \mid x \in \mathbf{x}, y \in \mathbf{y}\}, \forall \diamond \in \{+, -, \times, \div\}$. Thus, the ranges of the four elementary interval arithmetic operations are exactly the ranges of the their real-valued counterparts. Although this rule characterizes these operations mathematically, interval arithmetic's usefulness is due to the *operational definitions* based on interval bounds [11]. For example, let $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$, we define

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

Moreover, if such operations are composed, *bounds on the ranges* of factorable real functions can be obtained.

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. Readers are referred to [12, 13, 11, 2] for more details on basic interval methods.

2.2 Interval Constraint Propagation

The *tree representation* of constraint systems has been proposed in Benhamou *et al.* [4], therein each factorable constraint $r(t_1, \ldots, t_k)$ is represented by an *attribute tree* whose root node represents the k-ary relation symbol r, and the terms t_i are composed of nodes representing either a variable, a constant, or an elementary operation. Moreover, each node but the root is associated with two intervals, one for forward evaluation and the other for backward propagation.

The constraint propagation algorithm HC4 in [4], also referred to as the forward-backward contractor (see [2]), is based on the following two main processes. The first one is the *forward evaluation* which is recursively performed by a post-order traversal of the tree representation from leaves to roots in order to evaluate the ranges of sub-expressions represented by the tree nodes using *natural interval extension.* The second one is the *backward propagation* on the tree representation which is recursively performed by a pre-order traversal of the tree representation of each constraint from root to leaves in order to prune the corresponding interval associated with each node of the tree using the *projection narrowing operator* associated with the father of the node. Readers are referred to [4] for more details.

3 Numerical Constraint Propagation on DAGs

We will consider a constraint system of the form (1); the constraints can be equations or inequalities depending on whether the corresponding components of $\mathbf{b}$, called *constraint ranges*, are thin intervals (i.e. of form $[b_i, b_i]$).

Example 1. Consider the following parametric constraint system

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \leq 7, \\ x^2\sqrt{y} - 2xy + 3\sqrt{y} \in [p, q], \\ x \in [1, 16], \ \ y \in [1, 16]. \end{cases} \tag{2}$$

The first constraint is an inequality with constraint range $[-\infty, 7]$. The second constraint can be either an equation or an inequality depending on the parameters (p, q). For instance, the second constraint is an equation if $(p, q) = (0, 0)$ and an inequality if $(p, q) = (0, 2)$. Throughout this paper, we will use $(p, q) = (0, 2)$.

3.1 DAG Representation

We assume that readers are already familiar with fundamental concepts in graph theory like *directed acyclic graph/multigraph.* In the representation of the NCSPs we will use *directed acyclic multigraph with ordered edges* (for the definition readers are referred to [1] and references therein); for short, this is a directed acyclic multigraph, in which the incoming and outgoing edges at every node are totally ordered.

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

We use a directed acyclic multigraph, whose edges are totally ordered, together with an ordering on the vertices, as obtained in Theorem 1, to represent the constraint system (1), for short we call it a *Directed Acyclic Graph* (DAG). In that *DAG representation*, every node represents an elementary operation such as $+$, $\times$, $\div$, log, exp, ... and every edge represents the computational flow associated with a coefficient. In practice, we have to use multigraphs instead of simple graphs for the representation because some special operations can take the same input more than once, for example, when the expression x^x is represented by the elementary power operation x^y. The ordering of edges is needed for non-commutative operations like the division, but not for commutative operations. For convenience, a virtual ground node is added to the DAG to be the parent of all the nodes representing the constraints. In fact, the ground node can be interpreted as the logical 'AND' operation. Each node $\mathbf{N}$ in the DAG is associated with an interval, denoted $\mathbb{I}(\mathbf{N})$, in which the exact range of the corresponding sub-expression must lie.

Example 2. The DAG representation of (2) is depicted in Figure 1. The sequence of nodes $\{\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_{10}\}$ is an ordering of the nodes that satisfies Theorem 1.

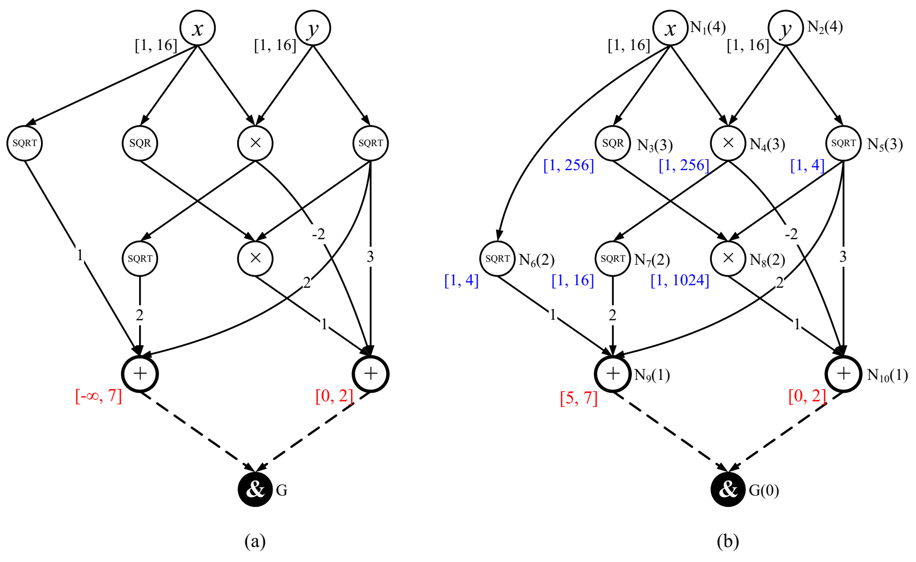

Fig. 1. The DAG representation (a) before and (b) after performing node ordering and recursive forward evaluation

3.2 Forward Evaluation and Backward Propagation on DAGs

In practice, we often see functions of form $f : D \rightarrow \mathbb{R}^m$, where $D \subset \mathbb{R}^n$. Quite often, in range analysis we need f to accept input from the domain $\mathbb{R}^n$, so we have to find a proper way to extend functions in a consistent way.

Definition 1 (Extended Function). *Let $f : D \rightarrow \mathbb{R}^m$ be a function, where $D \subseteq \mathbb{R}^n$, and S a subset of $2^{\mathbb{R}}$, the power set of $\mathbb{R}$. A function $g : \mathbb{R}^n \rightarrow \mathbb{R}^m \cup S^m$ is called an S-*extended function *of f if*

$$g(x) = \begin{cases} f(x) & \textit{if } x \in D, \\ y \in S^m & \textit{otherwise} \end{cases} \tag{3}$$

It is to easy see that there is only one S-extended function if S has only one element, for instance, when S is either $\{\emptyset\}$ or $\{\mathbb{R}\}$.

Example 3. The domain of the standard division x/y is $D_{\div} = \{(x, y) \in \mathbb{R}^2 \mid y \neq 0\}$. The unique $\{\emptyset\}$-extended function of the standard division is defined by

$$x \div_{\emptyset} y = \begin{cases} x/y & \text{if } y \neq 0, \\ \emptyset & \text{otherwise} \end{cases} \tag{4}$$

The unique $\{\mathbb{R}\}$-extended function of the standard division is defined by

$$x \div_{\mathbb{R}} y = \begin{cases} x/y & \text{if } y \neq 0, \\ \mathbb{R} & \text{otherwise} \end{cases} \tag{5}$$

The following is a $\{\emptyset, \mathbb{R}\}$-extended function of the standard division:

$$x \div_\star y = \begin{cases} x/y & \text{if } y \neq 0, \\ \emptyset & \text{if } x \neq 0, y = 0, \\ \mathbb{R} & \text{otherwise} \end{cases} \tag{6}$$

In the next definition, we extend the concept of inclusion function of [2].

Definition 2 (Inclusion Function). *Let S be a subset of $2^{\mathbb{R}}$, and $f : \mathbb{R}^n \to \mathbb{R}^m \cup S^m$ an S-extended function of a function $\varphi : D \to \mathbb{R}^m$, where $D \subseteq \mathbb{R}^n$. A function $[f] : \mathbb{I}^n \to \mathbb{I}^m$ is called an* inclusion function *of f and of φ if $\forall \mathbf{x} \in \mathbb{I}^n : f(\mathbf{x}) \subseteq [f](\mathbf{x})$, where $f(\mathbf{x}) = \{f(x) \mid x \in \mathbf{x} \cap D\} \cup_{x \in \mathbf{x} \setminus D} f(x)$.*[3]

Example 4. Let $\mathbf{x} = [\underline{x}, \overline{x}], \mathbf{y} = [\underline{y}, \overline{y}]$. We give as example three natural inclusion functions for the divisions defined by (4), (5) and (6) respectively.

$$\mathbf{x}[\div_\emptyset]\mathbf{y} = \begin{cases} \emptyset & \text{if } \mathbf{y} = [0,0], \\ [0,0] & \text{else if } \mathbf{x} = [0,0], \\ \mathbf{x} \div \mathbf{y} & \text{else if } 0 \notin \mathbf{y}, \\ [\underline{x}/\overline{y}, +\infty] & \text{else if } \underline{x} \geq 0 \wedge \underline{y} = 0, \\ [-\infty, \underline{x}/\underline{y}] & \text{else if } \underline{x} \geq 0 \wedge \overline{y} = 0, \\ [-\infty, \overline{x}/\overline{y}] & \text{else if } \overline{x} \leq 0 \wedge \underline{y} = 0, \\ [\overline{x}/\underline{y}, +\infty] & \text{else if } \overline{x} \leq 0 \wedge \overline{y} = 0, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{7}$$

$$\mathbf{x}[\div_\mathbb{R}]\mathbf{y} = \begin{cases} \mathbf{x} \div \mathbf{y} & \text{if } 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{8}$$

$$\mathbf{x}[\div_\star]\mathbf{y} = \begin{cases} \mathbf{x}[\div_\emptyset]\mathbf{y} & \text{if } 0 \notin \mathbf{x} \vee 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{9}$$

It is easy to see that $\forall \mathbf{x}, \mathbf{y} \in \mathbb{I} : \mathbf{x}[\div_\emptyset]\mathbf{y} \subseteq \mathbf{x}[\div_\star]\mathbf{y} \subseteq \mathbf{x}[\div_\mathbb{R}]\mathbf{y}$. Unfortunately, some interval implementations use the division $[\div_\mathbb{R}]$, while it is safe to use the division $[\div_\emptyset]$ in some computations such as forward evaluation, as described hereafter.

The *natural inclusion function* of f (see [2]), denoted by $\mathbf{f}$, is an example of an inclusion function, where in the factorable form of f each real variable is replaced by an interval variable and each operation is replaced by its interval counterpart.

In the DAG representation of (1), let $\mathbf{N}$ be a node which is not the ground node and has k children $\{\mathbf{C}_i\}_{i=1}^k$. The elementary operation represented by $\mathbf{N}$ is a function $f : D_f \to \mathbb{R}$, where $D_f \subseteq \mathbb{R}^k$. Hence, the relationship between $\mathbf{N}$ and its children can be written as $\mathbf{N} = f(\mathbf{C}_1, \ldots, \mathbf{C}_k)$.[4] Let $[f]$ be an inclusion function of the $\{\emptyset\}$-extended function of f. The *forward evaluation at node* $\mathbf{N}$ *using the inclusion function* $[f]$ is defined as follows

$$\texttt{FE}(\mathbf{N}, [f]) \equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap [f](\mathbb{I}(\mathbf{C}_1), \ldots, \mathbb{I}(\mathbf{C}_k))\} \tag{10}$$

[3] The set union of vectors is performed in component-wise fashion.

[4] Where we abuse the notation of a node for the real variable represented by it.

The aim of forward evaluation is to evaluate the ranges of nodes based on their children's ranges.

The aim of the *backward propagation* is to prune the intervals associated with children based on the constraint range of their parent. In other words, for each child $\mathbf{C}_i$ the backward propagation evaluates the i-th projection of the relation $\mathbf{N} = f(\mathbf{C}_1, \ldots, \mathbf{C}_k)$ on the variable represented by $\mathbf{C}_i$. It is then called the i-th backward propagation at $\mathbf{N}$ and denoted by $\mathtt{BP}(\mathbf{N}, \mathbf{C}_i)$. For convenience, we define the following sequence as the backward propagation at node $\mathbf{N}$

$$\mathtt{BP}(\mathbf{N}) = \{\mathtt{BP}(\mathbf{N}, \mathbf{C}_i)\}_{i=1}^{k} \tag{11}$$

Although the projection of relations is expensive in general, an evaluation of the projection of elementary operations can be obtained at low cost. Indeed, suppose that from the relation $\mathbf{N} = f(\mathbf{C}_1, \ldots, \mathbf{C}_k)$ we can infer an equivalent relation $\mathbf{C}_i = g_i(\mathbf{N}, \{\mathbf{C}_j\}_{j=1;j\neq i}^{k})$ for some $i \in \{1, \ldots, k\}$, where g_i is a function from $\mathbb{R}^k$ to $\mathbb{R}$. Let $[g_i]$ be an inclusion function of g_i. The i-th backward propagation can then be obtained as follows

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \equiv \{\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap [g_i](\mathbb{I}(\mathbf{N}), \{\mathbb{I}(\mathbf{C}_j)\}_{j=1;j\neq i}^{k})\} \tag{12}$$

In case that we cannot infer such a function g_i, more complicated rules to obtain the i-th projection of the relation $\mathbf{N} = f(\mathbf{C}_1, \ldots, \mathbf{C}_k)$ have to be constructed if the cost is low, alternatively the relation can be ignored. Fortunately, we can evaluate those projections for most elementary operations at low cost.

Example 5. Let f be the elementary operation represented by $\mathbf{N}$. We will use the notation $\oslash$ to mean that either the division $[\div_\star]$ or the division $[\div_\mathbb{R}]$ can be used at the place the notation $\oslash$ appears. The rules for the forward evaluation and the backward propagation are given as follows:

– if f is a univariate function such as sqr, sqrt, exp, log,... we can define

$$\mathtt{FE}(\mathbf{N}, [f]) \equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap [f](\mathbb{I}(\mathbf{C}_1))\}$$
$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_1) \equiv \{\mathbb{I}(\mathbf{C}_1) := \mathbb{I}(\mathbf{C}_1) \cap [f^{-1}](\mathbb{I}(\mathbf{N}))\}\ (f^{-1}(.) \text{ is the pre-image})$$

– if f is defined by $f(x_1, \ldots, x_k) = \alpha + \sum_{i=1}^{k} \alpha_i x_i$, we define

$$\mathtt{FE}(\mathbf{N}, \mathbf{f}) \equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap (\alpha + \sum_{i=1}^{k} \alpha_i \mathbb{I}(\mathbf{C}_i))\}$$

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \equiv \{\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap \frac{1}{\alpha_i}(\mathbb{I}(\mathbf{N}) - \alpha - \sum_{j=1;j\neq i}^{k} \alpha_j \mathbb{I}(\mathbf{C}_j))\}\ (i = 1, ..., k)$$

– if f is defined by $f(x_1, \ldots, x_k) = \alpha \prod_{i=1}^{k} x_i$, we define

$$\mathtt{FE}(\mathbf{N}, \mathbf{f}) \equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap \alpha \prod_{i=1}^{k} \mathbb{I}(\mathbf{C}_i)\}$$

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \equiv \{\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap (\mathbb{I}(\mathbf{N}) \oslash (\alpha \prod_{j=1;j\neq i}^{k} \mathbb{I}(\mathbf{C}_j)))\}\ (i = 1, \ldots, k)$$

– if f is defined by $f(x, y) = x/y$, i.e. $k = 2$, we define

$$\begin{aligned}\texttt{FE}(\mathbf{N}, \mathbf{f}) &\equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap \mathbf{f}(\mathbb{I}(\mathbf{C}_1), \mathbb{I}(\mathbf{C}_2))\}, \text{ where } \mathbf{f} \in \{[\div_\emptyset], [\div_\star], [\div_\mathbb{R}]\}\\ \texttt{BP}(\mathbf{N}, \mathbf{C}_1) &\equiv \{\mathbb{I}(\mathbf{C}_1) := \mathbb{I}(\mathbf{C}_1) \cap (\mathbb{I}(\mathbf{N}) \times \mathbb{I}(\mathbf{C}_2))\}\\ \texttt{BP}(\mathbf{N}, \mathbf{C}_2) &\equiv \{\mathbb{I}(\mathbf{C}_2) := \mathbb{I}(\mathbf{C}_2) \cap (\mathbb{I}(\mathbf{C}_1) \oslash \mathbb{I}(\mathbf{N}))\}\end{aligned}$$

4 Coordinating Constraint Propagation and Search

We focus on solving techniques following the *branch-and-prune* framework, where the solving process is performed by repeatedly interleaving *pruning* techniques, which use local techniques such as constraint propagation to reduce the variable domains, with *branching* techniques, which split a problem into subproblems. Each subproblem constructed in the solving process usually consists of a subset of constraints, hereafter called *active constraints*, which need to be satisfied, and sub-domains of the initial variable domains. Therefore, solving techniques that use the DAG representation need to create such a representation for each subproblem. The simplest way is to build a new DAG to represent each subproblem. However, the total cost of creating such DAGs for the whole solving process is probably high. As an alternative, we propose in Section 4.1 to modify a piece of information attached to the initial DAG in order to make the initial DAG interpreted as the DAG representation of a subproblem without the necessity of creating new DAGs.

4.1 Partial Forward-Backward Propagation on DAGs

Partial DAG Representation. In order to represent the set of active constraints without having to create new DAGs, we use a vector, V_{oc}, whose size is equal to the number of nodes of the DAG representing the initial problem. For each node $\mathbf{N}$ of the DAG, we use the entry $V_{oc}[\mathbf{N}]$ to count the number of occurrences of $\mathbf{N}$ in the factorable form of the active constraints. In Figure 2, we give a recursive procedure, called `NodeOccurrences`, to compute such a vector. It is easy to see that $V_{oc}[\mathbf{N}] = 0$ if and only if $\mathbf{N}$ is not in the representation of the active constraints. Therefore, by combining the initial DAG with the vector V_{oc}, we have a so-called *partial DAG representation* for each subproblem. In the latter computations, we can use the partial DAG representation in a way similar to using the (full) DAG representation, except that we ignore all nodes corresponding to zeros of the vector V_{oc}. An example of the partial DAG representation for the problem (2) is depicted in Figure 3.

Forward-Backward Propagation on DAGs. Inspired by the original forward evaluation and backward propagation in [4], we devise a new algorithm for numerical constraint propagation, that is based on partial DAG representation instead of the tree representation. We call the new algorithm *"Forward-Backward Propagation on DAG"* and denote it by `FBPD` (pronounced *For-Bap-Dag*). In Figure 4, we present the main steps of `FBPD`. In the next paragraphs, we describe in detail the procedures that are not made explicit in Figure 4.

```
procedure NodeOccurrences(in : N; out : $V_{oc}$)
    for each child C of node N do
        $V_{oc}[\mathbf{C}] := V_{oc}[\mathbf{C}] + 1$;
        NodeOccurrences($\mathbf{C}, V_{oc}$);
    end-for
end
```

Fig. 2. If traversing all active constraints, the `NodeOccurrences` procedure counts the number of occurrences of each node in the factorable form of the active constraints

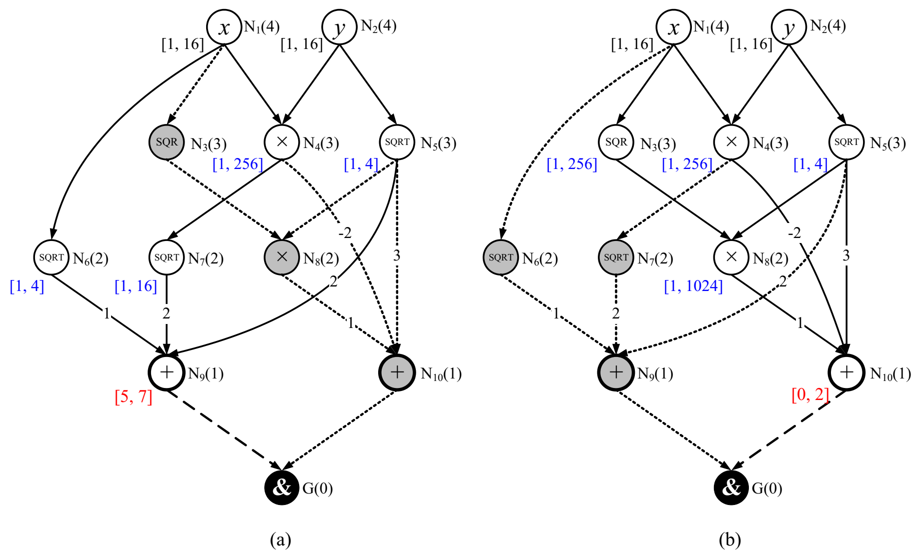

Fig. 3. The partial DAG representation of the problem (2) when (a) the first constraint, or (b) the second constraint is the unique active constraint. The grey nodes are not counted, hence are ignored in computations. The dotted edges are redundant. The node levels are not updated

Recursive Forward Evaluation. Similar to the `HC4` algorithm, we perform a recursive forward evaluation at the initialization phase (lines 01-08) to evaluate the ranges of the nodes in the partial DAG representation. In Figure 5, we give the details of a procedure, named `ForwardEvaluation`, for such a recursive evaluation. The results of the recursive forward evaluation of (2) are depicted in Figure 1b and Figure 3 for the case that both constraints are active and the case that only one constraint is active, respectively.

Get the Next Node for Further Processing. Like with the `HC4` algorithm [4], in the main body of the `FBPD` algorithm there are two principal processes: forward evaluation and backward propagation. However, unlike the `HC4` algorithm, the `FBPD` algorithm performs these processes for a single node instead of all the nodes at once. Therefore, in the `FBPD` algorithm, the choice of the next node for further processing can be made adaptively based on the results of the previous

```
/* $D(\mathbf{G})$ : a DAG with the ground $\mathbf{G}$; $\mathcal{C}$ : active constraints; $\mathcal{D}$: variable domains */
algorithm FBPD(in : $D(\mathbf{G}), \mathcal{C}$; in/out : $\mathcal{D}$)
01:  Reset the node ranges of $D(\mathbf{G})$ to the ranges in either $\mathcal{D}$, $\mathcal{C}$, or $[-\infty, +\infty]$;
02:  $\mathcal{L}_f := \emptyset$; $\mathcal{L}_b := \emptyset$; $V_{oc} := (0, \ldots, 0)$; $V_{ch} := (0, \ldots, 0)$;
03:  $V_{lvl} := (0, \ldots, 0)$; /* this can be made optional together with line 06 */
04:  for each node $\mathbf{C}$ representing an active constraint in $\mathcal{C}$ do
05:      NodeOccurrences($\mathbf{C}, V_{oc}$);
06:      NodeLevel($\mathbf{C}, V_{lvl}$); /* this can be made optional */
07:      ForwardEvaluation($\mathbf{C}, V_{ch}, \mathcal{L}_b$);
08:  end-for
09:  while $\mathcal{L}_b \neq \emptyset \vee \mathcal{L}_f \neq \emptyset$ do
10:      $\mathbf{N}$ := getNextNode($\mathcal{L}_b, \mathcal{L}_f$);
11:      if $\mathbb{I}(\mathbf{N})$ was taken from $\mathcal{L}_b$ then
12:          for each child $\mathbf{C}$ of $\mathbf{N}$ do
13:              BP($\mathbf{N}, \mathbf{C}$); /* see the description of (12) */
14:              if $\mathbb{I}(\mathbf{C}) = \emptyset$ then return infeasible;
15:              if $\mathbb{I}(\mathbf{C})$ changed enough for forward evaluation then
16:                  for each $\mathbf{P} \in parents(\mathbf{C}) \setminus \{\mathbf{N}, \mathbf{G}\}$ do
17:                      if $V_{oc}[\mathbf{P}] > 0$ then put $\mathbf{P}$ into $\mathcal{L}_f$;
18:              end-if
19:              if $\mathbb{I}(\mathbf{C})$ changed enough for backward propagation then
20:                  Put $\mathbf{C}$ into $\mathcal{L}_b$;
21:          end-for
22:      else /* $\mathbf{N}$ was taken from $\mathcal{L}_f$ */
23:          FE($\mathbf{N}, [f]$); /* $f$ is the operator at $\mathbf{N}$, see the description of (10) */
24:          if $\mathbb{I}(\mathbf{N}) = \emptyset$ then return infeasible;
25:          if $\mathbb{I}(\mathbf{N})$ changed enough for forward evaluation then
26:              for each $\mathbf{P} \in parents(\mathbf{N}) \setminus \{\mathbf{G}\}$ do
27:                  if $V_{oc}[\mathbf{P}] > 0$ then put $\mathbf{P}$ into $\mathcal{L}_f$;
28:          end-if
29:          if $\mathbb{I}(\mathbf{N})$ changed enough for backward propagation then
30:              Put $\mathbf{N}$ into $\mathcal{L}_b$;
31:      end-if
32:  end-while
33:  Update $\mathcal{D}$ with the ranges of the nodes representing the variables;
end
```

Fig. 4. The Partial Forward-Backward Propagation on DAG (FBPD) algorithm

```
procedure ForwardEvaluation(in : $\mathbf{N}$; in/out : $V_{ch}, \mathcal{L}_b$)
    if $\mathbf{N}$ is a leaf or $V_{ch}[\mathbf{N}] = 1$ then return;
    for each child $\mathbf{C}$ of node $\mathbf{N}$ do ForwardEvaluation($\mathbf{C}, V_{ch}, \mathcal{L}_b$);
    if $\mathbf{N} = \mathbf{G}$ then return;
    FE($\mathbf{N}, [f]$); $V_{ch}[\mathbf{N}] := 1$; /* $f$ is the operator at $\mathbf{N}$, similar to line 23 in Figure 4*/
    if $\mathbb{I}(\mathbf{N}) = \emptyset$ then return infeasible;
    if $\mathbb{I}(\mathbf{N})$ changed enough for backward propagation then put $\mathbf{C}$ into $\mathcal{L}_b$;
end
```

Fig. 5. This is a procedure to do a recursive forward evaluation

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

Fig. 6. This is a procedure assigning a node level to each node in a DAG.

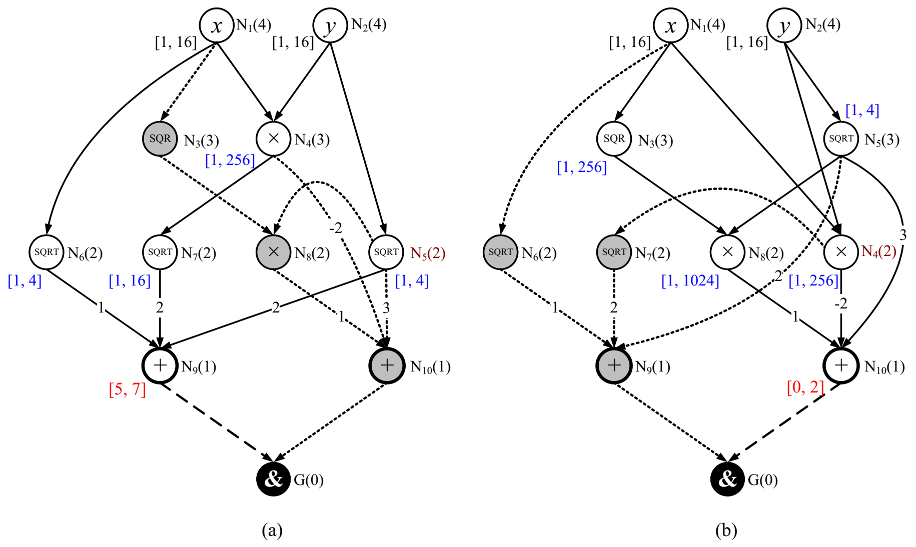

Fig. 7. The node levels are updated at each call to the FBPD algorithm

processes. The algorithm uses two waiting lists to store the nodes waiting for further processing. The first list, $\mathcal{L}_f$, is a list of nodes that is scheduled for forward evaluation, that is, for evaluating its range based on its children's ranges. The second list, $\mathcal{L}_b$, is a list of nodes that is waiting for backward propagation, that is, for pruning its children's ranges based on its range. In general, when $\mathcal{L}_f$ contains many nodes, the nodes should be sorted such that the forward evaluation of a node is performed after the forward evaluation of its children. Analogously, the nodes in $\mathcal{L}_b$ should be sorted such that the backward propagation at a node is performed before the backward propagation at its children. The `NodeLevel` procedure in Figure 6 assigns to each node a *node level* such that the node level of an arbitrary node is smaller than the node levels of its descendants. We then sort the nodes of $\mathcal{L}_b$ and $\mathcal{L}_f$ in ascending order and descending order of node levels, respectively, to meet the above requirements. The `getNextNode` function at line 10 in Figure 4 chooses one of the two nodes at the beginning of $\mathcal{L}_b$ and $\mathcal{L}_f$. The strategy that we use in our implementation is "backward propagation first", that is, taking the node at the beginning of $\mathcal{L}_b$ whenever it is available.

The call to the `NodeLevel` procedure at line 06 in Figure 4 can be made optional as follows. The first option allows to invoke `NodeLevel` only at the first call to `FBPD`. The node levels of the initial DAG still meet the requirements on the ordering of the waiting lists. The numbers in brackets next to the node names in Figure 3 are the node levels computed for the initial DAG. Figure 7 illustrates the second option that allows to invoke `NodeLevel` at line 06 in Figure 4 every time `FBPD` is invoked.

When Are the Changes of Node Ranges Enough? For simplicity, in Figure 4 (lines 15, 19, 25, 29) we only briefly present the procedures to check whether the node ranges have been changed enough for further processing. Hereafter, we will detail them. Let denote by $\mathbf{M}$ the node $\mathbf{C}$ at line 13 or the node $\mathbf{N}$ at line 23. In Figure 4, the forward evaluation at line 23 and the backward propagation at line 13 are of form

$$\mathbb{I}(\mathbf{M}) := \mathbb{I}(\mathbf{M}) \cap \mathbf{y} \tag{13}$$

where $\mathbf{y}$ is the interval computed by the forward evaluation or backward propagation before intersecting with $\mathbb{I}(\mathbf{M})$.

Let W_{old} and W_{new} be the widths of $\mathbb{I}(\mathbf{M})$ and $\mathbb{I}(\mathbf{M}) \cap \mathbf{y}$, respectively. In practice, the change of $\mathbb{I}(\mathbf{M})$ after performing (13) is considered enough for doing the forward evaluation at its parents if the conditions $W_{new} < r_f W_{old}$ and $W_{new} + d_f \le W_{old}$ hold, where r_f is a real number in $(0, 1)$ and d_f is a small positive real number. The numbers r_f and d_f can be predefined or dynamically computed. Similarly, the change of $\mathbb{I}(\mathbf{M})$ after performing (13) is considered enough for doing the backward propagation at $\mathbf{M}$ if the conditions $W_{new} < r_b W_{old}$ and $W_{new} + d_b \le W_{old}$ hold, where r_b is a real number in $(0, 1)$ and d_b is a small positive real number. Moreover, if $\mathbf{y}$ is computed by the forward evaluation (at line 23), the additional condition $\mathbf{y} \not\subseteq \mathbb{I}(\mathbf{M})$ must also hold.

4.2 Combining Propagation and Search Using a DAG

The most common framework for solving NCSPs is the branch-and-prune framework. The most common exhaustive search is the bisection. Bisection search is suitable for problems with isolated solutions. However, it is often inefficient for problems with a continuum of solutions. Therefore, for the problems with a continuum of solutions we need more advanced search techniques like `UCA5`, `UCA6` and `UCA6`+ (see [6, 7]). They all can be seen as instances of the generic branch-and-prune search described in Figure 8. In general, the search scheme produces two lists. The first list, $\mathcal{L}_\forall$, consists of feasible sub-domains. The second list, $\mathcal{L}_\varepsilon$, consists of tuples of tiny sub-domains, that are smaller than the required resolution ε, and the sets of constraints, that are still active in the corresponding sub-domains.

5 Experiments

We have carried out experiments on `FBPD` and two other recent interval propagation techniques. The first one is an implementation of Box Consistency [14, 15] in a commercial product named ILOG Solver (v6.0, 11/2003), hereafter denoted

```
algorithm BnPSearch(in : $\mathcal{V}, \mathcal{C}, \mathcal{D}$; out : $\mathcal{L}_\forall, \mathcal{L}_\varepsilon$)
    Construct a DAG, $D(\mathbf{G})$, for the initial problem $(\mathcal{V}, \mathcal{C}, \mathcal{D})$;
    FPBD($D(\mathbf{G}), \mathcal{C}, \mathcal{D}$); /* Prune the domains in $\mathcal{D}$ */
    if $infeasible$ is detected then return $infeasible$;
    if domains in $\mathcal{D}$ are small enough then $\mathcal{L}_\varepsilon := \mathcal{L}_\varepsilon \cup \{(\mathcal{C}, \mathcal{D})\}$; return;
    $\mathcal{L} := \{(\mathcal{C}, \mathcal{D})\}$;
    while $\mathcal{L} \neq \emptyset$ do
        Get a couple $(\mathcal{C}_0, \mathcal{D}_0)$ from $\mathcal{L}$;
        Split the problem $(\mathcal{V}, \mathcal{C}_0, \mathcal{D}_0)$ into subproblems $\{(\mathcal{V}, \mathcal{C}_i, \mathcal{D}_i)\}_{i=1}^{k}$; where $\mathcal{C}_i \subseteq \mathcal{C}_0$
        for $i := 1$ to $k$ do
            FPBD($D(\mathbf{G}), \mathcal{C}_i, \mathcal{D}_i$); /* Prune the domains in $\mathcal{D}_i$ */
            if $infeasible$ is detected then continue for;
            if $\mathcal{C}_i = \emptyset$ then $\mathcal{L}_\forall := \mathcal{L}_\forall \cup \{\mathcal{D}_i\}$; continue for;
            if domains in $\mathcal{D}_i$ are small enough then
                $\mathcal{L}_\varepsilon := \mathcal{L}_\varepsilon \cup \{(\mathcal{C}_i, \mathcal{D}_i)\}$
            else
                $\mathcal{L} := \mathcal{L} \cup \{(\mathcal{C}_i, \mathcal{D}_i)\}$;
            end-if
        end-for
    end-while
end
```

Fig. 8. A generic branch-and-prune search using FPBD for pruning.

by BOX. The second one is called HC4 (Revised Hull Consistency) from [4]. The experiments are carried out on 33 problems which are unbiasedly selected and divided into five test cases:

– The test case T_1 consists of 8 problems with isolated solutions that are solvable by all three propagators.
– The test case T_2 consists of 4 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 the three techniques to stop due to timeout or due to running more than 10^6 iterations.
– The test case T_4 consists of 7 small problems with a continuum of solutions that are solvable at the predefined resolution 10^{-2}.
– The test case T_5 consists of 6 hard problems with a continuum of solutions that are solvable at the predefined resolution 10^{-1}.

The timeout value is set to **10 hours** for all the test cases, except that the timeout value used for T_3 is set to **2 hours** due to lack of time for this version.[5] The timeout values will be used as the running time for the techniques which are 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

[5] We will give the test results for timeout 10 hours in an extended version elsewhere.

technique, called `UCA6`, for inequalities (see [6, 7]). 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 `FBPD` 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 `FBPD` 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 `FBPD` 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 $\sqrt[d]{V/D}$; where d is the dimension of the problem, 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 `FBPD` 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 Tables 1 and 2. In Table 3, we give the *overrun ratio* of each propagator for the test case T_1. The overrun ratio is defined by $\varepsilon / \sqrt[d]{V/N}$; where ε is the required resolution, d is the dimension of the problem, V is the total volume of the output boxes, N is the number of output boxes. Clearly, `FBPD` outperforms both `BOX` and `HC4` by an order of magnitude or more in speed, while being roughly the same quality w.r.t. enclosure properties in case where the solution set to be enclosed by boxes of macroscopic size (i.e. for continuum of solutions). For isolated solutions, very narrow boxes are produced by any technique in comparison to the required resolution. However, the new technique is 1.1-2 times less tight than the other techniques in the measure on reduction per dimension (which hardly matters in applications). In comparison with `HC4`, a constraint propagation technique that is similar to `FBPD` but works on the tree representation instead of DAGs, `FBPD` is clearly more suitable for applications.

6 Conclusion

We propose a constraint propagation technique, `FBPD`, which makes the fundamental framework for constraint propagation on DAGs [1] efficient and practical, and a method to coordinate constraint propagation and exhaustive search using a single DAG for each problem. The experiments carried out on various problems show that the new approach outperforms previously available propagation techniques by an order of magnitude or more in speed, while being roughly the same quality w.r.t. enclosure properties.

In other views, `FBPD` can be seen as a special instance of the generic combination scheme, `CIRD`, proposed by Vu *et al.* [16]. Therefore, combining and unifying the strength of `FBPD` and `CIRD1`, an instance of the `CIRD` scheme, to solve problems with either isolated or non-isolated solutions is straightforward.

Table 1. (*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.

Propagator	(*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
`FBPD`	**1.000**	1.000	1.000	1.000	**1.000**	0.922	1.000	1.000
`BOX`	17.800	**0.625**	**0.342**	**0.731**	20.919	**0.944**	**0.873**	**0.854**
`HC4`	181.469	0.906	1.266	0.988	403.915	0.941	0.896	0.879

Table 2. 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
`FBPD`	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
`BOX`	24.21	28.98	5.79	11.55	31.85
`HC4`	94.42	691.24	13.63	191.86	651.31

Table 3. This table contains the overrun ratios for the test case T_1. An overrun ratio greater than 1 would suffice for applications.

Problem →	BIF-3	REI-3	WIN-3	ECO-5	ECO-6	NEU-6	ECO-7	ECO-8	**Average**
`FBPD`	1.626	1.360	2.075	1.711	1.676	3.198	1.513	1.455	**1.880**
`BOX`	2.957	1.974	3.080	1.579	1.660	6.748	1.521	1.485	**2.625**
`HC4`	2.229	1.914	1.492	1.647	1.679	4.949	1.488	1.449	**2.106**

Acknowledgements

This research was 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 used in the COCONUT project, and thank IRIN team at University of Nantes for the `HC4` code. We also thank Prof. Arnold Neumaier at University of Vienna (Austria) for fruitful discussions and very valuable input.

References

1. Schichl, H., Neumaier, A.: Interval Analysis on Directed Acyclic Graphs for Global Optimization (2004) preprint - University of Vienna, Autria.

2. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. First edn. Springer (2001)
3. Granvilliers, L., Goualard, F., Benhamou, F.: Box Consistency through Weak Box Consistency. In: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'99). (1999) 373–380
4. 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
5. Benhamou, F., Goualard, F.: Universally Quantified Interval Constraints. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000). (2000) 67–82
6. 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)
7. 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
8. Kearfott, B.: Interval Computations: Introduction, Uses, and Resources. Euromath Bulletin **2(1)** (1996) 95–112
9. Burkill, J.: Functions of Intervals. Proceedings of the London Mathematical Society **22** (1924) 375–446
10. Moore, R.: Interval Arithmetic and Automatic Error Analysis in Digital Computing. PhD thesis, Stanford University, USA (1962)
11. Hickey, T., Ju, Q., Van Emden, M.: Interval Arithmetic: from Principles to Implementation. Journal of the ACM (JACM) **48(5)** (2001) 1038–1068
12. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic Press, New York, NY (1983)
13. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge Univ. Press, Cambridge (1990)
14. Benhamou, F., McAllester, D., Van Hentenryck, P.: CLP(Intervals) Revisited. In: Proceedings of the International Logic Programming Symposium. (1994) 109–123
15. Van Hentenryck, P., McAllester, D., Kapur, D.: Solving Polynomial Systems Using a Branch and Prune Approach. SIAM Journal of Numerical Analysis **34(2)** (1997)
16. Vu, X.H., Sam-Haroud, D., Faltings, B.: A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation. Technical Report IC/2004/39, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland (2004)