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

Xuan-Ha Vu
Artificial Intelligence Laboratory,
Swiss Federal Institute
of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
xuan-ha.vu@epfl.ch

Hermann Schichl
Faculty of Mathematics,
University of Vienna,
Nordbergstr. 15, A-1090 Wien,
Austria
hermann.schichl@esi.ac.at

Djamila Sam-Haroud
Artificial Intelligence Laboratory,
Swiss Federal Institute
of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
jamila.sam@epfl.ch

***Abstract*— The foundational paper of H. SCHICHL and A. NEUMAIER [1] has given the fundamentals of interval analysis on *directed acyclic graphs* (DAGs) for global optimization and constraint propagation. We show in this paper how constraint propagation on DAGs can be made efficient and practical by: (i) working on *partial DAG representations*; and (ii) enabling the flexible choice of the interval inclusion functions during propagation. We then propose a new simple algorithm which coordinates constraint propagation and exhaustive search for solving numerical constraint satisfaction problems. The experiments carried out on different 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.**

I. INTRODUCTION

Many real world problems require solving numerical constraint satisfaction problems (NCSPs). An NCSP is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ which consists of a finite set $\mathcal{V}$ of variables taking their values in domains $\mathcal{D}$ over real numbers 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 solutions is called the solution set.

In practice, numerical constraints are often equalities or inequalities expressed in *factorable* form (that is, they can be recursively composed of elementary functions such as $+$, $-$, $\times$, $\div$, $\log$, $\exp$, sqr, $\sin$, $\cos$, $\ldots$). In other words, such an NCSP can be expressed as

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

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 two interval vectors of sizes n and m respectively.

Many solution techniques have been proposed in *Constraint Programming* and *Mathematical Programming* to solve NCSPs. To achieve full mathematical rigor when dealing with floating-point numbers, most solution techniques have been based on *interval arithmetic* or its variants. During the last ten years, a lot of work has been done to devise *inclusion tests* and *contractors* by using interval arithmetic (see the book by JAULIN *et al.* [2]). The role of an inclusion test is to check whether the domain of a variable is included in the solution set. A contractor, also called a *narrowing operator* [3], [4] or *contracting operator* [5], [6], [7], is a method that reduces variable domains such that no solution is lost. Various basic inclusion tests and contractors have been described in [2]. In particular, an interesting approach called *interval constraint propagation* [4], [8], [9] was developed, which associates *constraint propagation*/*local consistency* techniques, as defined in artificial intelligence, with interval analytic methods. Advanced contractors, such as the *forward-backward contractor* [2], [4], result from the interval constraint propagation approach. In brief, the forward-backward contractor, which is first introduced as `HC4` in [4], is a method to propagate domain reductions forwards and backwards through the trees which represent the composition of constraints. The method is therefore referred to as *forward-backward propagation* in this paper. More recently, a fundamental framework for interval analysis on directed acyclic graphs (DAGs) has been proposed by SCHICHL & NEUMAIER [1], which showed that the forward-backward propagation can also be performed on DAGs. Replacing trees by DAGs potentially reduces the number of computations in the forward-backward propagation.

In practice, inclusion tests and contractors are interleaved with *exhaustive search* to compute a representation of the solution set. Search by *bisection* is the most commonly used technique. However, advanced algorithms [6], [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).

The contribution of this paper is twofold. Firstly, we show how the framework proposed by SCHICHL & NEUMAIER [1], can be made efficient and practical for performing constraint propagation on DAGs (Section III). Secondly, we propose a new algorithm to coordinate constraint propagation and exhaustive search on DAGs (Section IV). More precisely, we propose a technique for performing forward-backward

propagation on DAGs that is able to work on *partial DAG representations*. The algorithm restricts the work to relevant *subsets of constraints* while keeping the *initial* DAG representation for the problem. The other specificity of our forward-backward propagation technique is that it makes it possible to flexibly choose different *inclusion functions*[1] at different stages of the propagation. We then propose a solving technique which coordinates our partial forward-backward propagation on DAGs with exhaustive search, in a branch-and-prune framework. 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 for unbiasedly chosen benchmarks (Section V).

II. Background and Notation

We start by presenting the necessary background and fundamental notations.

A. Fundamental Notations

The power set of a set A is denoted by 2^A, that is, $2^A \equiv \{S \mid S \subseteq A\}$. The set of real numbers is denoted by $\mathbb{R}$. The set of floating-point numbers is denoted by $\mathbb{F}$.

B. Factorable Functions

Hereafter, we recall the *factorable function* concept, with slight modifications, that appeared in [11], [12].

Definition 1 (Factorable Function): A function is called a factorable function using a finite set, E, of elementary operations if it is a recursive composition of operations in E, variables, and constants.

Example 1: The function $f(x,y) = \sin x + 2xy$ is a factorable function using elementary operations in $\{+, \times, \sin\}$. The recursive composition is given as follows.

$$\begin{aligned} f &= f_1 + f_2 \\ f_1 &= \sin x \\ f_2 &= 2 \times f_3 \\ f_3 &= x \times y \end{aligned}$$

C. 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 a survey paper by R. B. Kearfott [13], a form of interval arithmetic perhaps first appeared in [14]. Modern interval arithmetic was developed independently in late 1950s by several researchers, including M. Warmus [15], T. Sunaga [16], and R. E. Moore [17], with Moore finally setting the firm foundation for the field in his many publications, including the foundational book [18]. Since then, interval arithmetic has been used to solve numerical problems with guaranteed rigor. Fundamentally, if $\mathbf{x}$ and $\mathbf{y}$ are two real intervals, then the four elementary operations for *idealized interval arithmetic* obey the rule

$$\mathbf{x} \diamond \mathbf{y} = \{x \diamond y \mid x \in \mathbf{x}, y \in \mathbf{y}\}, \forall \diamond \in \{+, -, \times, \div\}. \quad (2)$$

Thus, the results of the four elementary interval arithmetic operations are exactly the ranges of their real-valued counterparts. Although the rule (2) characterizes these operations mathematically, the usefulness of interval arithmetic is due to the *operational definitions* based on interval bounds [19]. For example, let $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$, interval arithmetic shows

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

Elementary operations $\psi : D \subseteq \mathbb{R} \to \mathbb{R}$ can also be extended to intervals, and usually this is done by defining

$$\psi(\mathbf{x}) = \{\psi(x) \mid x \in \mathbf{x}\}, \quad (3)$$

whenever $\mathbf{x} \subseteq D$. For a deeper discussion of elementary functions, especially for the cases where $D \neq \mathbb{R}$, see Section III-B. Moreover, if such operations and elementary functions are composed, *bounds on the ranges* of factorable real functions can be obtained.

The finite nature of computers precludes an exact representation of the real numbers. 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* [20]. The set of real intervals is then approximated by the set $\mathbb{I}$ of intervals with bounds in $\mathbb{F}_\infty$. The power of interval arithmetic lies in its implementation on computers. In particular, *outwardly rounded* interval arithmetic allows *rigorous enclosures* for the ranges of operations and functions. This makes a qualitative difference in scientific computations, since the results are now intervals in which the exact result must lie. Interval arithmetic can be carried out for virtually any expression that can be evaluated in floating-point arithmetic. Readers are referred to [2], [19], [10], [21] for more details on basic interval methods.

The Cartesian product of intervals is called *interval box*, or *box* for short. An interval is said to be *canonical* if its bounds are equal or adjacent in $\mathbb{F}_\infty$. A box is said to be *canonical* if all of its intervals are canonical.

D. Interval Constraint Propagation

1) Tree Representation: The *tree representation* of constraint systems has been proposed by 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 exact range of the corresponding expression at a node must lie in the intervals associated with

[1] Inclusion function is a well-known concept in interval analysis [2], [10] for enclosing the ranges of real-valued functions

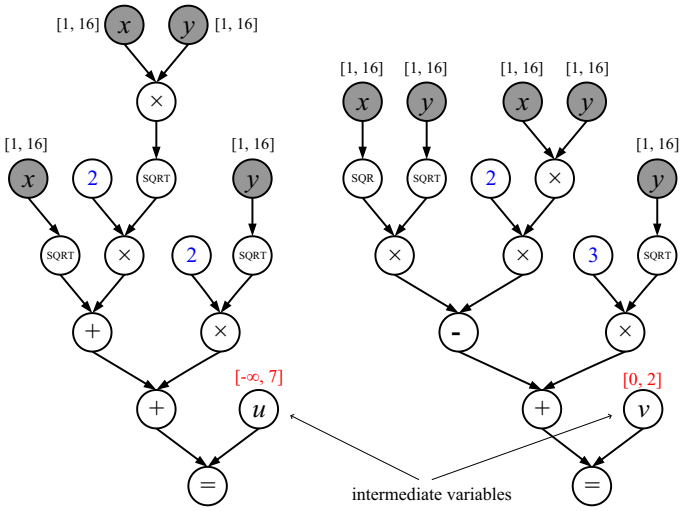

Fig. 1. This is the tree representation of the problem in Example 3. The two variables, x and y, are represented by grey nodes which are pointers to two domains of the two variables. The roots of the trees representing the constraints. Each node representing a real constant is associated with the smallest interval that contains the constant. If not specified, the domains of other nodes are initialized to $[-\infty, +\infty]$

the node. In order to represent the inequalities of (1) without introducing special root nodes, intermediate variables are used to represent the *constraint ranges* (i.e. the components of $\mathbf{b}$ in (1)).

Example 2: The tree representation of the system (4) is depicted in Figure 1.

2) Forward Evaluation and Backward Propagation on Trees: The constraint propagation algorithm named `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 the so-called *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 by using the *projection narrowing operator* associated with the father of the node. Readers are referred to [4] for more details.

III. Numerical Constraint Propagation on DAGs

In this section, we show how the forward-backward propagation defined in [4], that works on tree representations, can be extended to work on DAGs. We start by recalling the basic concepts on DAGs (Section III-A). We notably show that several inclusion functions can be flexibly chosen during the forward-backward propagation (Section III-B), which might improve the tightening of the variable domains as shown in Section III-C. We finally present in detail how the forward-backward propagation can be performed on DAGs (Section III-C).

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

Example 3: Consider the following parametric constraint system

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \leq 7, \\ x^2\sqrt{y} - 2xy + 3\sqrt{y} \in [p, q], \\ x \in [1, 16],\ y \in [1, 16]. \end{cases} \tag{4}$$

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

A. DAG Representation

For completeness, we recall hereafter some fundamental concepts in graph theory related to the representation of a constraint system, which was proposed in [1].

Definition 2 (Directed Multigraph): A *directed multigraph* (V, E, f) consists of a finite set V of *vertices* (also called *nodes*), a finite set E of *edges*, and a mapping $f : E \to V \times V$ such that $\forall e \in E : f_s(e) \neq f_t(e)$, where $f = (f_s, f_t)$. For every edge $e \in E$ we define the *source* of e as $f_s(e)$ and the *target* of e as $f_t(e)$.

Definition 3 (Directed Multigraph with Ordered Edges): A *directed multigraph with ordered edges* is a quadruple $(V, E, f, \preceq)$ such that (V, E, f) is a directed multigraph and $(E, \preceq)$ is a totally ordered set.

Definition 4 (Directed Path): Let $\mathcal{G} = (V, E, f)$ be a directed multigraph. A *directed path* from $v_1 \in V$ to $v_2 \in V$ is a sequence, $\{e_i\}_{i=1}^n$, of edges such that $v_1 = f_s(e_1)$, $v_2 = f_t(e_n)$, and $\forall i \in \{1, \ldots, n-1\} : f_t(e_i) = f_s(e_{i+1})$. The directed path is called a *cycle* if $v_1 = v_2$. $\mathcal{G}$ is called *acyclic* if it does not contain a cycle.

Definition 5: Let (V, E, f) be a directed multigraph. For any two vertices $v_1, v_2 \in V$ we say that v_1 is a *parent* of v_2 and v_2 is a *child* of v_1 if $\exists e \in E : f_s(e) = v_2 \wedge f_t(e) = v_1$. We call v_1 an *ancestor* of v_2 and v_2 a *descendant* of v_1 if there exists a directed path from v_2 to v_1.

Theorem 1: For every directed acyclic multigraph (V, E, f) there exists a total order $\preceq$ on 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 the *DAG representation*, every node represents a variable or an elementary operation (such as $+$, $\times$, $\div$, $\log$, $\exp$, $\ldots$) 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 power operation x^y, thus, we do not need a new univariate operation

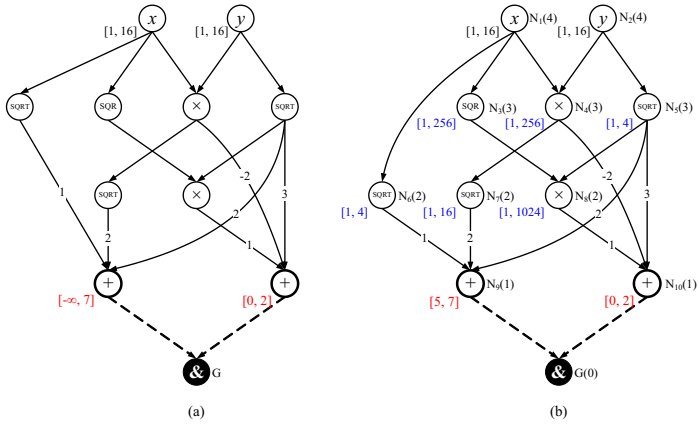

Fig. 2. The DAG representation (a) before and (b) after performing node ordering and recursive forward evaluation

for x^x. The ordering of edges is needed for non-commutative operations like the division, but not for commutative operations. For convenience, a virtual ground node, $\mathbf{G}$, 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 associated sub-expression must lie.

Example 4: The DAG representation of (4) is depicted in Figure 2. The sequence of nodes $\{\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_{10}\}$ is an ordering of the nodes which satisfies Theorem 1.

B. Extended Functions

In practice, we often see functions of the form $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$. For example, the division by zero is not defined. As a consequence, in standard interval arithmetic the division of two intervals is not defined if the denominator contains zero. In such cases, many implementations of interval arithmetic give, by convention, the interval $[-\infty, +\infty]$ as result. If we use these implementations to evaluate the ranges of functions, we usually get unnecessarily overestimated ranges such as $[-\infty, +\infty]$. In order to avoid such overestimations, we have to extend functions in a consistent way for use in different computations which use inclusion functions. Hereafter, we give a way to extend functions which are only defined on subsets of $\mathbb{R}^n$.

Definition 6 (Extended Function): Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a function and S a subset of $2^{\mathbb{R}}$. A function $g : \mathbb{R}^n \to \mathbb{R}^m \cup S^m$ is called an *S-extended function* of f if

$$g(x) = \begin{cases} f(x) & \text{if } x \in D, \\ y \in S^m & \text{otherwise} \end{cases} \tag{5}$$

It is easy to 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 5: 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 as

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

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

$$x \div_{\mathbb{R}} y = \begin{cases} x/y & \text{if } y \neq 0, \\ \mathbb{R} & \text{otherwise} \end{cases} \tag{7}$$

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

$$x \div_{\star} y = \begin{cases} x/y & \text{if } y \neq 0, \\ \emptyset & \text{if } x \neq 0, y = 0, \\ \mathbb{R} & \text{otherwise} \end{cases} \tag{8}$$

In the next definition, we extend the inclusion function concept of [2] by using the notion of extended-function to guarantee the consistency and correctness.

Definition 7 (Inclusion Function): Let S be a subset of $2^{\mathbb{R}}$, and $g : \mathbb{R}^n \to \mathbb{R}^m \cup S^m$ an S-extended function of a function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$. A function $[g] : \mathbb{I}^n \to \mathbb{I}^m$ is called an *inclusion function* of g (and then of f) if the *inclusion property* holds, that is,

$$\forall \mathbf{x} \in \mathbb{I}^n : g(\mathbf{x}) \subseteq [g](\mathbf{x}),$$

where[2]

$$g(\mathbf{x}) \equiv \{g(x) \mid x \in \mathbf{x} \cap D\} \cup \bigcup_{x \in \mathbf{x} \setminus D} g(x).$$

The *natural inclusion function* of f (see [2]), denoted by $\mathbf{f}$, is an instance of inclusion functions which is constructed as follows: in the factorable form of f each real variable (resp. constant) is replaced by an interval variable (resp. constant) and each operation is replaced by its interval counterpart.

Example 6: 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 (6), (7) and (8), respectively.

$$\mathbf{x}[\div_{\emptyset}]\mathbf{y} = \begin{cases} \emptyset & \text{if } \mathbf{y} = [0, 0], \\ [0, 0] & \text{else if } \mathbf{x} = [0, 0], \\ \mathbf{x} \div \mathbf{y} & \text{else if } 0 \notin \mathbf{y}, \\ [\underline{x}/\overline{y}, +\infty] & \text{else if } \underline{x} \geq 0 \wedge \underline{y} = 0, \\ [-\infty, \underline{x}/\underline{y}] & \text{else if } \underline{x} \geq 0 \wedge \overline{y} = 0, \\ [-\infty, \overline{x}/\overline{y}] & \text{else if } \overline{x} \leq 0 \wedge \underline{y} = 0, \\ [\overline{x}/\underline{y}, +\infty] & \text{else if } \overline{x} \leq 0 \wedge \overline{y} = 0, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{9}$$

$$\mathbf{x}[\div_{\mathbb{R}}]\mathbf{y} = \begin{cases} \mathbf{x} \div \mathbf{y} & \text{if } 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{10}$$

$$\mathbf{x}[\div_{\star}]\mathbf{y} = \begin{cases} \mathbf{x}[\div_{\emptyset}]\mathbf{y} & \text{if } 0 \notin \mathbf{x} \vee 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise} \end{cases} \tag{11}$$

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 and use the division $[\div_{\star}]$ in some computations such as backward propagation,

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

as described in Section III-C. In some interval libraries the extended interval division as defined in [22] is implemented, which is the tightest all-purpose interval division, but $[\div_\emptyset]$ and $[\div_\star]$ both provide a bit better bounds than that for forward and backward propagation, respectively.

C. Forward Evaluation and Backward Propagation on DAGs

As in [4], the aim of the *forward evaluation* phase is to evaluate the range of a node based on the ranges of its children. The *backward evaluation* phase is concerned with pruning the intervals associated with the children based on that of the considered node. The existing forward-backward propagation scheme in [4] only allows using natural inclusion functions for both phases. We show how this propagation can be enhanced by enabling the use of several types of inclusion functions at different propagation phases.

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 $f : D_f \subseteq \mathbb{R}^k \to \mathbb{R}$. Hence, the relationship between $\mathbf{N}$ and its children can be written as $\mathbf{N} = f(\mathbf{C}_1, \ldots, \mathbf{C}_k)$.[3] 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

$$\texttt{FE}(\mathbf{N}, [f]) \equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap [f](\mathbb{I}(\mathbf{C}_1), \ldots, \mathbb{I}(\mathbf{C}_k))\}. \quad (12)$$

This forward evaluation computes the range of a node based on the ranges of its children by using an inclusion function of the elementary operation represented by this node. For example, considering the node $\mathbf{N}_7$ in Figure 2, we can use any inclusion function of the $\{\emptyset\}$-extended function of the square root operation. In the implementation, we use the natural inclusion for simplicity.

Remark 1: We can also replace the inclusion function $[f]$ in (12) by an inclusion function of the recursive subexpression whose variables are user's ones. For instance, we can replace $[f]$ of the node $\mathbf{N}_7$ by the natural inclusion function of the recursive subexpression composed of the nodes in $\{\mathbf{N}_7, \mathbf{N}_4, \mathbf{N}_1, \mathbf{N}_2\}$, that is, by the bivariate interval function $\sqrt{\mathbf{x}\mathbf{y}}$.

The *backward propagation* prunes the intervals associated with children based on the constraint range of their parents. 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 $\texttt{BP}(\mathbf{N}, \mathbf{C}_i)$. For convenience, we define the following sequence as the backward propagation at node $\mathbf{N}$

$$\texttt{BP}(\mathbf{N}) = \{\texttt{BP}(\mathbf{N}, \mathbf{C}_i)\}_{i=1}^k. \quad (13)$$

Although the exact projection of relations is expensive in general, an evaluation of the exact 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, denoted $\texttt{BP}(\mathbf{N}, \mathbf{C}_i)$, can then be defined as

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

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.

Definition 8: Let f be the elementary operation represented by $\mathbf{N}$ as discussed above. 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, but the former is better. The rules for the forward evaluation and the backward propagation are given as follows:

1) If f is a univariate function such as sqr, sqrt, $\exp, \log, \ldots$ and $[f]$ is defined as in (12); we define

$$\begin{aligned} \texttt{FE}(\mathbf{N}, [f]) &\equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap [f](\mathbb{I}(\mathbf{C}_1))\}, \\ \texttt{BP}(\mathbf{N}, \mathbf{C}_1) &\equiv \{\mathbb{I}(\mathbf{C}_1) := \mathbb{I}(\mathbf{C}_1) \cap [f^{-1}](\mathbb{I}(\mathbf{N}))\}, \end{aligned}$$

   where we abuse the notation of inclusion function, $[f^{-1}](\mathbf{x})$, to denote some intervals containing the pre-image $f^{-1}(\mathbf{x})$.

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

$$\begin{aligned} \texttt{FE}(\mathbf{N}, \mathbf{f}) &\equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap (\alpha + \sum_{i=1}^k \alpha_i \mathbb{I}(\mathbf{C}_i))\}, \\ \texttt{BP}(\mathbf{N}, \mathbf{C}_i) &\equiv \{\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap \frac{1}{\alpha_i}(\mathbb{I}(\mathbf{N}) - \alpha - \\ &\qquad \sum_{j=1;j\neq i}^k \alpha_j \mathbb{I}(\mathbf{C}_j))\} \ (i = 1, \ldots, k). \end{aligned}$$

3) If f is defined as $f(x_1, \ldots, x_k) = \alpha \prod_{i=1}^k x_i$, we define

$$\begin{aligned} \texttt{FE}(\mathbf{N}, \mathbf{f}) &\equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap \alpha \prod_{i=1}^k \mathbb{I}(\mathbf{C}_i)\}, \\ \texttt{BP}(\mathbf{N}, \mathbf{C}_i) &\equiv \{\mathbb{I}(\mathbf{C}_i) := \mathbb{I}(\mathbf{C}_i) \cap (\mathbb{I}(\mathbf{N}) \oslash \\ &\qquad (\alpha \prod_{j=1;j\neq i}^k \mathbb{I}(\mathbf{C}_j)))\} \ (i = 1, \ldots, k). \end{aligned}$$

4) If f is defined as $f(x, y) = x/y$, i.e. $k = 2$, we define

$$\begin{aligned} \texttt{FE}(\mathbf{N}, \mathbf{f}) &\equiv \{\mathbb{I}(\mathbf{N}) := \mathbb{I}(\mathbf{N}) \cap \mathbf{f}(\mathbb{I}(\mathbf{C}_1), \mathbb{I}(\mathbf{C}_2))\}, \\ \texttt{BP}(\mathbf{N}, \mathbf{C}_1) &\equiv \{\mathbb{I}(\mathbf{C}_1) := \mathbb{I}(\mathbf{C}_1) \cap (\mathbb{I}(\mathbf{N}) \times \mathbb{I}(\mathbf{C}_2))\}, \\ \texttt{BP}(\mathbf{N}, \mathbf{C}_2) &\equiv \{\mathbb{I}(\mathbf{C}_2) := \mathbb{I}(\mathbf{C}_2) \cap (\mathbb{I}(\mathbf{C}_1) \oslash \mathbb{I}(\mathbf{N}))\}, \end{aligned}$$

   where $\mathbf{f} \in \{[\div_\emptyset], [\div_\star], [\div_{\mathbb{R}}]\}$.

[3] In this paper, we abuse the notation of a node for the real variable represented by it.

Proposition 1: The forward evaluation and backward propagation rules given in Definition 8 never discard a solution of the problem represented by the corresponding DAG.

Proof: The proof directly follows the definitions in Section III-B and Section III-C, and is therefore omitted for simplicity. ■

IV. Coordinating Propagation and Search

We now tackle the issue of coordinating constraint propagation and search for solving NCSPs. It builds on the classical *branch-and-prune* framework, where the solving process is performed by repeatedly interleaving a *pruning* step with a *branching* step. The former uses local techniques such as constraint propagation to reduce the variable domains, while the latter splits a problem into subproblems.

At each branching step, a subproblem has to be solved which consists of a subset of the original constraints called the set of *active constraints*[4] in this paper. The active constraints are defined on sub-domains of the initial variable domains.

If the pruning technique uses the DAG representation, the DAG representation needs to be constructed for each subproblem. The simplest way, therefore, consists of explicitly building a new DAG to represent each subproblem considered. However, since there is often a huge number of branching steps during a complete solving process, the total cost of creating such DAGs is potentially high.

As an alternative, we propose 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. Using this information, it becomes possible to perform forward-backward evaluation on *partial DAG representations* of the original problem without increasing much the time and space needed.

In Section IV-A, we present how partial forward-backward propagations can be performed on partial DAG representations of the original problem. We then devise in Section IV-B a detailed search algorithm based on the partial forward-backward propagation on DAGs.

A. Partial Forward-Backward Propagation on DAGs

1) 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 3, 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 (4) is depicted in Figure 4.

[4]Note that this notion differs from the meaning *active constraint* in the optimization literature.

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

Fig. 3. If traversing all active constraints, the `NodeOccurrences` procedure will count the number of occurrences of each node in the factorable form of the active constraints

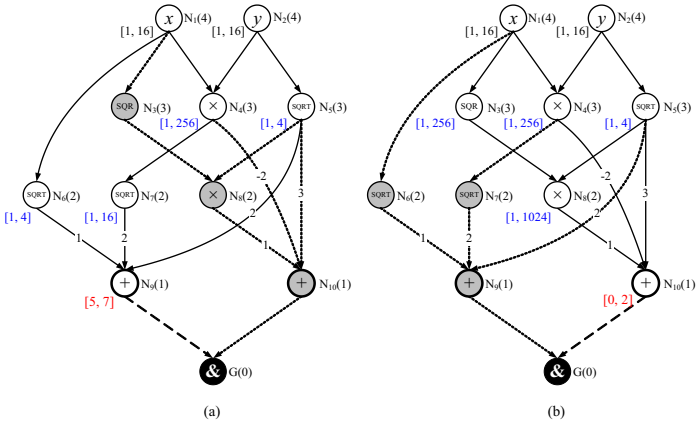

Fig. 4. The partial DAG representation of the problem (4) when (a) the first constraint, or (b) the second constraint is the unique active constraint. The grey nodes are not counted, hence are ignored in computations. The dotted edges are redundant. The node levels are not updated

2) Partial Forward-Backward Propagation on Initial DAG: Inspired by the original forward evaluation and backward propagation in [4], we devise a new algorithm for numerical constraint propagation, that is based on the partial DAG representation instead of the tree representation. We call the new algorithm *"Forward-Backward Propagation on a DAG"* and denote it by `FBPD`. In Figure 5, we present the main steps of `FBPD`. 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 adaptively made based on the results of the previous processes. Moreover, in the `FBPD` algorithm, the choice of the inclusion function $[f]$ to be used in the forward evaluation and the backward propagation is not necessarily fixed. In the next paragraphs, we describe in detail the procedures that are not made explicit in Figure 5.

a) 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 6, we give the details of a procedure, named `ForwardEvaluation`, for

```
/* D(G) : a DAG with the ground G */
/* 𝒟: variable domains; 𝒞 : active constraints */
algorithm FBPD(in : D(G), 𝒞; in/out : 𝒟)
00:   Reset all node ranges of D(G) to [−∞, +∞];
01:   Set the node ranges of vars & constraints to 𝒟 & 𝒞, resp.;
02:   ℒ_f := ∅; ℒ_b := ∅; V_oc := (0, ..., 0); V_ch := (0, ..., 0);
03:   V_lvl := (0, ..., 0); /* can be made optional with line 06 */
04:   for each node C representing an active constraint in 𝒞 do
05:       NodeOccurrences(C, V_oc);
06:       NodeLevel(C, V_lvl); /* this can be made optional */
07:       ForwardEvaluation(C, V_ch, ℒ_b);
08:   end-for
09:   while ℒ_b ≠ ∅ ∨ ℒ_f ≠ ∅ do
10:       N := getNextNode(ℒ_b, ℒ_f);
11:       if 𝕀(N) was taken from ℒ_b then
12:           for each child C of N do
13:               BP(N, C); /* see Definition 8 */
14:               if 𝕀(C) = ∅ then return infeasible;
15:               if 𝕀(C) changed enough for doing FE(.) then
16:                   for each P ∈ parents(C) \ {N, G} do
17:                       if V_oc[P] > 0 then put P into ℒ_f;
18:                   end-for
19:               end-if
20:               if 𝕀(C) changed enough for doing BP(.) then
21:                   Put C into ℒ_b;
22:           end-for
23:       else /* N was taken from ℒ_f */
24:           FE(N, [f]); /* f is the operator at N, see (12) */
25:           if 𝕀(N) = ∅ then return infeasible;
26:           if 𝕀(N) changed enough for doing FE(.) then
27:               for each P ∈ parents(N) \ {G} do
28:                   if V_oc[P] > 0 then put P into ℒ_f;
29:               end-for
30:           end-if
31:           if 𝕀(N) changed enough for doing BP(.) then
32:               Put N into ℒ_b;
33:       end-if
34:   end-while
35:   Update 𝒟 with the ranges of the nodes of the variables;
end
```

Fig. 5. FBPD – the partial Forward-Backward Propagation on DAG algorithm

```
procedure ForwardEvaluation(in : N; in/out : V_ch, ℒ_b)
    if N is a leaf or V_ch[N] = 1 then return;
    for each C ∈ children(N) do
        ForwardEvaluation(C, V_ch, ℒ_b);
    end-for
    if N = G then return;
    FE(N, [f]); /* similar to line 24 in Figure 5 */
    V_ch[N] := 1; /* the range of this node is cached */
    if 𝕀(N) = ∅ then return infeasible;
    if 𝕀(N) changed enough for backward propagation then
        Put C into ℒ_b;
    end-if
end
```

Fig. 6. This is a procedure to do a recursive forward evaluation

such a recursive evaluation. To avoid evaluating the same sub-expressions many times, we use a vector, V_{ch}, to mark the caching status of nodes. The results of the recursive forward evaluation of (4) are depicted in Figure 2b and Figure 4 for the case that both constraints are active and the case that only one constraint is active, respectively.

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

Fig. 7. This is a procedure assigning a node level to each node in a DAG.

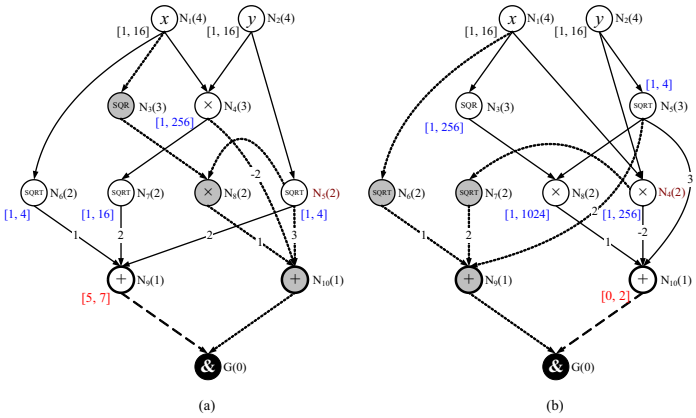

Fig. 8. The node levels are updated at each call to the FBPD algorithm

b) Get the Next Node for Further Processing: The FBPD 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 7 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 call to the NodeLevel procedure at line 06 in Figure 5 can be made optional as follows. The first option allows invoking 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 4 are the node levels computed for the initial DAG. Figure 8 illustrates the second option that allows invoking NodeLevel at line 06 in Figure 5 every time FBPD is invoked.

The getNextNode function at line 10 in Figure 5 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 $\mathcal{L}_b$ is not empty. Of course, other selection strategies can also be used.

c) When Are the Changes of Node Ranges Enough?: For simplicity, in Figure 5 (lines 15, 20, 26, 31) 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 $\mathbf{M}$ denote the node $\mathbf{C}$ at line 13 or the node $\mathbf{N}$ at line 24. In Figure 5, the forward evaluation at line 24 and the backward propagation at line 13 are of form

$$\mathbb{I}(\mathbf{M}) := \mathbb{I}(\mathbf{M}) \cap \mathbf{y}, \tag{14}$$

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 (14) is considered enough for doing the forward evaluation at its parents if the conditions $W_{new} < r_f W_{old}$ and $W_{new} + d_f \leq W_{old}$ 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 (14) is considered enough for doing the backward propagation at $\mathbf{M}$ if the conditions $W_{new} < r_b W_{old}$ and $W_{new} + d_b \leq W_{old}$ 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 24), the additional condition $\mathbf{y} \nsubseteq \mathbb{I}(\mathbf{M})$ must also hold.

The FBPD algorithm is *contractive* and *correct* in the following meaning.

Proposition 2: We define a function $F : \mathbb{I}^n \times 2^{\mathbb{R}^n} \rightarrow \mathbb{I}^n$ to represent the FBPD algorithm. This function takes as input the variable domains (in 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 FBPD algorithm. If the input problem contains only the elementary operations f defined in Definition 8, then the FBPD algorithm terminates at a finite number of iterations and the following properties hold:

$$\begin{array}{lllll} (i) & F(\mathbf{B}, S) & \subseteq & \mathbf{B} & \text{(Contractiveness)} \\ (ii) & F(\mathbf{B}, S) & \supseteq & \mathbf{B} \cap S & \text{(Correctness)} \end{array}$$

Proof: All the ranges of nodes in the DAG representation of the problem are never inflated at each step of the FBPD algorithm, then the FBPD algorithm must terminate at a finite number of iterations due to the finite nature of floating-point numbers. In particular, the ranges of the nodes representing the variables are never inflated, hence, the property (i) holds. Moreover, the forward evaluations and backward propagations used in the FBPD algorithm are defined in Definition 8, they never discard a solution (due to the inclusion property of inclusion functions). Therefore, the property (ii) holds. ■

B. Combining Propagation and Search Using a DAG

Branch-and-Prune is the most common framework for exhaustively solving NCSPs. The most widely used algorithm for search is bisection, hence called the *bisection search*. It is suitable for problems with isolated solutions. However, it is often inefficient for problems with a continuum of solutions, for instance, problems with inequalities. Therefore, for problems with a continuum of solutions we need more advanced search techniques like UCA6, UCA6 and UCA6+ (see [6], [7]). They all can be viewed as instances of the generic branch-and-prune search described in Figure 9. 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, which are boxes either smaller than the required resolution ε or canonical, and the sets of constraints, that are still active in the corresponding sub-domains.

```
algorithm BnPSearch(in : V, D, C; out : L∀, Lε)
    Construct a DAG, D(G), for the initial problem (V, D, C);
    FPBD(D(G), C, D); /* Prune the domains in D */
    if infeasible is detected then return infeasible;
    if domains in D are small enough then
        Lε := Lε ∪ {(D, C)};
        return;
    end-if
    L := {(D, C)};
    while L ≠ ∅ do
        Get a couple (D0, C0) from L;
        Split the problem (V, D0, C0) into subproblems
            {(V, D1, C1), ..., (V, Dk, Ck)}; where C_{i=1,k} ⊆ C0
        for i := 1 to k do
            FPBD(D(G), Ci, Di); /* Prune the domains in Di */
            if infeasible is detected then continue for;
            if Ci = ∅ then
                L∀ := L∀ ∪ {Di};
                continue for;
            end-if
            if domains in Di are small enough then
                Lε := Lε ∪ {(Di, Ci)}
            else
                L := L ∪ {(Di, Ci)};
            end-if
        end-for
    end-while
end
```

Fig. 9. A generic branch-and-prune search using FBPD for pruning.

It is easy to prove that, due to the finite nature of floating-point numbers, the branch-and-prune search presented in Figure 9 can obtain a predefined positive resolution ε, i.e. $\mathcal{L}$ becomes empty, after a finite number of steps. Moreover, the branch-and-prune search is a complete search technique because the FBPD algorithm is complete.

V. EXPERIMENTS

We have carried out experiments on the FBPD algorithm and two other well-known state-of-the-art interval constraint processing techniques. The first one is an implementation of Box Consistency [23], [24] 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 [4]. The experiments are carried out on 33 problems which are *unbiasedly* chosen and divided into five test cases for analyzing the test results:[5]

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

- 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 FBPD 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 FBPD being stopped due to running more than 10^6 splittings; 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 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,[6] for inequalities (see [6], [7]). The comparison of the interval propagation techniques is based on the measures of

- *The running time:* The relative ratio of the running time of each propagator to that of FBPD 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 FBPD is called the *relative cluster ratio*.
- *The number of splittings/iterations:* 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 FBPD 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 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*.
- *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 I and Table II.

Note 1: 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 I, 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 I, the averages of the relative ratios are taken over all the problems in the test cases T_4, T_5.

In Table III, we give the *overrun ratio* of each propagator for the test case T_1. The overrun ratio is defined as $\varepsilon / \sqrt[d]{V/N}$; 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, at least for the unbiasedly chosen benchmarks, 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 about 1.1–2.0 times less tight than the other techniques in the measure on reduction per dimension (which hardly matters in applications). In comparison with HC4 (we recall that HC4 is 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.

TABLE I

THE COMPARISON OF THE THREE CONSTRAINT PROPAGATION TECHNIQUES IN SOLVING NCSPS.

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
FBPD	**1.000**	1.000	1.000	1.000	**1.000**	0.922	1.000	1.000
BOX	20.863	**0.625**	**0.342**	**0.731**	20.919	**0.944**	**0.873**	**0.854**
HC4	203.285	0.906	1.266	0.988	403.915	0.941	0.896	0.879

TABLE II

THE AVERAGES OF THE RELATIVE TIME RATIOS ARE TAKEN OVER THE PROBLEMS IN EACH TEST CASE.

Prop. ▼	(a) *Isolated Solutions*			(b) *Continuum of Solutions*	
	Case T_1	Case T_2	Case T_3	Case T_4	Case T_5
FBPD	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
BOX	24.21	28.98	13.45	11.55	31.85
HC4	94.42	691.24	68.17	191.86	651.31

TABLE III

THE OVERRUN RATIOS FOR THE TEST CASE T_1. (AN OVERRUN RATIO GREATER THAN 1 WOULD SUFFICE FOR APPLICATIONS.)

Prob. ▶	BIF3	REI3	WIN3	ECO5	ECO6	NEU6	ECO7	ECO8	**Average**
FBPD	1.626	1.360	2.075	1.711	1.676	3.198	1.513	1.455	**1.827**
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**

[6]The implementation of UCA6 is downloadable at http://www.mat.univie.ac.at/coconut-environment/ (in the BCS module) under the LGPL and GPL licenses.

VI. 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 for a set of unbiasedly chosen benchmarks, while being roughly the same quality w.r.t. enclosure properties. Moreover, the design nature of FBPD is similar to that of the HC4 algorithm. Therefore, we can use the FBPD algorithm in many applications and combination techniques which use the HC4 algorithm.

In other views, FBPD can be viewed as a special instance of a generic combination scheme, called CIRD, that was proposed by Vu *et al.* [25]. Moreover, our experiments show that the strengths of FBPD and CIRD[ai] (an instance of the CIRD scheme, that uses *affine arithmetic* and *interval arithmetic*) are complementary when considering problems have isolated or non-isolated solutions. Therefore, combining and unifying the strengths of FBPD and CIRD[ai] to solve problems with either isolated or non-isolated solutions is a straightforward direction in the near future.

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 the IRIN team at the University of Nantes for the HC4 code. We also thank Prof. Arnold Neumaier at the University of Vienna (Austria) for fruitful discussions and very valuable input.

References

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

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

[3] L. Granvilliers, F. Goualard, and F. Benhamou, "Box Consistency through Weak Box Consistency," in *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'99)*, November 1999, pp. 373–380.

[4] 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)*, Las Cruces, USA, 1999, pp. 230–244.

[5] F. Benhamou and F. Goualard, "Universally Quantified Interval Constraints," in *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000)*, 2000, pp. 67–82.

[6] 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.

[7] 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.

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

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

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

[11] G. P. McCormick, "Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems," *Mathematical Programming*, vol. 10, pp. 147–175, 1976.

[12] ——, *Nonlinear Programming: Theory, Algorithms and Applications*. John Wiley & Sons, 1983.

[13] R. B. Kearfott, "Interval Computations: Introduction, Uses, and Resources," *Euromath Bulletin*, vol. 2(1), pp. 95–112, 1996.

[14] J. C. Burkill, "Functions of Intervals," *Proceedings of the London Mathematical Society*, vol. 22, pp. 375–446, 1924.

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

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

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

[18] ——, *Interval Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1966.

[19] 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.

[20] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, March 1991.

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

[22] G. W. Walster, E. R. Hansen, and J. D. Pryce, "Extended Real Intervals and the Topological Closure of Extended Real Relations," Sun Microsystems, Tech. Rep., February 2000, http://wwws.sun.com/software/sundev/whitepapers/extended-real.pdf.

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

[24] 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.

[25] X.-H. Vu, D. Sam-Haroud, and B. Faltings, "Combining Multiple Inclusion Representations in Numerical Constraint Propagation," in *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. Florida, USA: IEEE Computer Society Press, November 2004.