Interval Propagation and Search on Directed Acyclic Graphs for Numerical Constraint Solving

Xuan-Ha Vu ha.vu@4c.ucc.ie
Cork Constraint Computation Centre,
University College Cork,
14 Washington Street West, Cork, Ireland

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

Djamila Sam-Haroud jamila.sam@epfl.ch
Artificial Intelligence Laboratory,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
Batiment IN, Station 14, CH-1015 Lausanne, Switzerland

Abstract

The fundamentals of interval analysis on *directed acyclic graphs* (DAGs) for global optimization and constraint propagation have recently been proposed by Schichl and Neumaier [2005]. For representing numerical problems, the authors use DAGs whose nodes are subexpressions and whose directed edges are computational flows. Compared to tree-based representations [Benhamou et al. 1999], DAGs offer the essential advantage of more accurately handling the influence of subexpressions shared by several constraints on the overall system during propagation.

In this paper we show how interval constraint propagation and search on DAGs can be made practical and efficient by : 1) flexibly *choosing the nodes* on which propagations must be performed, and 2) working with *partial* subgraphs of the initial DAG rather than with the entire graph.

We propose a new *interval constraint propagation* technique which exploits the influence of subexpressions on *all* the constraints together rather than on *individual* constraints. We then show how the new propagation technique can be integrated into *branch-and-prune* search to solve numerical constraint satisfaction problems.

This algorithm is able to outperform its obvious contenders, as shown by the experiments.[1]

1. Introduction

A *constraint satisfaction problem* (CSP) consists of a finite set of constraints specifying which combinations of values from given *domains* of variables are admitted. A CSP is said to be *numerical* if its domains are continuous. Numerical CSPs, such as systems of nonlinear equations and inequalities, arise in many applications and form a difficult problem class since

1. A short version of this paper has been published in [Vu et al. 2004b]

they are NP-hard. In practice, numerical constraints are usually expressed in *factorable form* which means that they are composed of elementary operators or functions (e.g., $+$, $*$, $\div$, $\sqrt{\ }$ and sin) and standard relations (e.g., $\leq$, $<$, $\neq$, $>$, and $\geq$). Many solution techniques exploit the factorability of such numerical constraints to efficiently solve numerical CSPs. To achieve full mathematical rigor when performing operations on floating-point numbers, most techniques are based on interval arithmetic or its variants.

The most commonly used complete strategy for finding the solutions of a numerical CSP is *branch-and-prune*, which interleaves branching steps with pruning steps. Roughly speaking, a *branching* step divides the problem into subproblems whose union is equivalent to the initial one in term of the solution set, and a *pruning* step reduces/simplifies the problem in some measure. The most well-known pruning technique is *domain reduction*, which reduces the domains of variables without discarding any solution of the problem.

Over the last twenty years, many domain reduction techniques based on *interval arithmetic* have been devised. In particular, an interesting approach in *constraint programming*, called *interval constraint propagation*, was developed in the 1990s (see [Benhamou and Older 1992, 1997], [Van Hentenryck 1997] and [Jaulin et al. 2001]). This approach combines constraint propagation techniques, as defined in artificial intelligence, with interval-analytic methods. The algorithm **HC4** [Benhamou et al. 1999] is one of the most prominent representatives of this family of domain reduction techniques. In **HC4**, each individual constraint is represented by a tree whose nodes and edges stand respectively for subexpressions and computational flows. Each node of the tree is associated with the (possible) range of the corresponding subexpression.

In order to reduce the variables' domains of a given constraint the technique recursively performs *forward evaluations* and then *backward projections* on the whole tree representing the constraint. These two steps compute the ranges of nodes based on the ranges of their children's and parents' respectively. When several constraints are involved, **HC4** performs forward evaluations and backward projections individually on each constraint, and then propagates the reduction of the variables' domains from tree to tree by using a variant of *arc consistency*, **AC3** [Mackworth 1977].

The fact that each constraint is propagated individually is one of the main limitations of this approach. The effects of the common subexpressions, shared by several constraints, is only roughly taken into account.

Recently, a fundamental framework for interval analysis on *directed acyclic graphs* (DAGs) has been proposed by Schichl and Neumaier [2005] which overcomes this limitation. The authors suggested to replace trees with DAGs and showed how to perform forward evaluations and backward projections using this particular representation. The shift to DAGs potentially reduces the amount of computation on common subexpressions shared by constraints, and explicitly relates constraints to constraints in the natural way they are composed, thus enhancing the constraint propagation process.

The constraint propagation technique proposed by [Schichl and Neumaier 2005] is a direct generalization of **HC4** in the sense that all the nodes of the DAG are forward evaluated then backward projected at once. In practice, and as the problems grow large, situations often occur where only a small number of nodes is worth considering for forward or backward inference as the other nodes leave the domain ranges unchanged after computation.

This paper builds on this idea and presents a new constraint propagation technique following the DAG-based framework [Schichl and Neumaier 2005]. The contribution is twofold. Firstly, we show how the DAG-based framework can be made efficient and practical by adaptively *performing forward evaluations and backward projections on chosen nodes* of a DAG (see Section 4, Section 4.3 and Section 5). In our approach, switching from evaluations to projections is made at the node level rather than at the tree or DAG level.

Secondly, we show how the new propagation technique can be integrated into a generic branch-and-prune search *without the necessity to create multiple DAGs* (see Section 4.3, Section 5 and Section 6). Our experiments carried out on impartially chosen benchmarks show that the new technique outperforms previously available propagation techniques by 1 to 2 orders of magnitude or more in speed, while being roughly the same quality with respect to enclosure properties (see Section 7).

The paper is organized as follows. Section 2 presents the necessary background and definitions, including the basic concepts of factorability (Section 2.1), interval arithmetic (Section 2.2), numerical constraint satisfaction (Section 2.3), and DAG representation of numerical CSPs (Section 2.4). Section 3 describes a slight modification to standard interval arithmetic that may reduce the amount of computation in constraint propagation. Forward evaluation and backward propagation on DAGs as well as the notion of *partial DAG representation* are presented in Section 4 and serve as basis for the core contributions of this paper (Sections 5 and 6). Finally, Section 7 discusses the preliminary experimental results.

2. Background and Definition

2.1 Factorable Form

In practice, most functions for modeling real-world applications can be expressed using elementary operations or functions such as $+$, $-$, $*$, $/$, sqr, exp, ln, and sin. If an expression is recursively composed of standard elementary operations and functions, it is called an *arithmetic expression* [Neumaier 1990, p. 13] or a *factorable expression* [McCormick 1976, 1983]. Factorable expressions play a significant role in algorithms for solving not only numerical CSPs but also other numerical problems such as optimization problems and automatic differentiation computations. For completeness, we recall in this section the concepts of *factorability*.

Notation 1 (Elementary Operations)**.** $\mathbb{E}_1$ *denotes the set of standard* elementary unary functions*, namely,* $\mathbb{E}_1 = \{\mathrm{abs}, \mathrm{sqr}, \mathrm{sqrt}, \exp, \ln, \sin, \cos, \arctan\}$. $\mathbb{E}_2$ *denotes the set of standard* elementary binary operations*, namely,* $\mathbb{E}_2 = \{+, -, *, /, \hat{}\}$.

In this paper, we extend the concept of an arithmetic expression to include other elementary operations.

Definition 2 (Factorable Expression)**.** *Let R be a nonempty set, $\{x_1, \dots, x_n\}$ a set of variables taking values in R, F a finite set of* elementary operations *of the form $f : R^k \to R$. An expression is said to be* factorable *in the (formal) variables $x_1, \dots, x_n$ using operations in F if it is a member of the minimal set $\mathcal{F} \equiv \mathcal{F}(R, F; x_1, \dots, x_n)$ satisfying the following* composition rules*:*

1. $R \subseteq \mathcal{F}$;

2. *$x_i \in \mathcal{F}$ for all $i = 1, \ldots, n$;*

3. *If $f : R^k \to R$ is in F and $e_1, \ldots, e_k \in \mathcal{F}$; then $f(e_1, \ldots, e_k) \in \mathcal{F}$.*

Notation 3. *We denote $\mathcal{E}(x_1, \ldots, x_n) \equiv \mathcal{F}(\mathbb{R}, \mathbb{E}_1 \cup \mathbb{E}_2; x_1, \ldots, x_n)$.*

If an expression E is factorable in variables $X \equiv \{x_1, \ldots, x_n\}$ using operations in F as in Definition 2 and if either $F = \mathbb{E}_1 \cup \mathbb{E}_2 \wedge R = \mathbb{R}$ holds or F is known from the context, then we say for short that E is factorable (in X).

The expression $f(x, y) = 2x^y + \sin x$ is factorable using the elementary operations in $\{+, *, \hat{}, \sin\}$. The composition is given as follows: $f_1 = x \hat{} y$ ($\equiv x^y$), $f_2 = 2 * f_1$, $f_3 = \sin(x)$, and $f = f_2 + f_3$. The expression $f(x, y)$ is also an arithmetic expression, namely, in $\mathcal{E}(x, y)$.

Definition 4 (Factorable Function)**.** *A function f is said to be* factorable *in variables $x_1, \ldots, x_n$ using the operations in a finite set F of elementary operations if it can be expressed by an expression that is factorable in variables $x_1, \ldots, x_n$ using elementary operations in F. If $F = \mathbb{E}_1 \cup \mathbb{E}_2$ or F is known from the context, we could just say for short that f is factorable (in variables $x_1, \ldots, x_n$).*

For example, the function $f(x, y) = 2x^y + \sin x$ is factorable using the operations in $\{+, *, \hat{}, \sin\}$, and is not factorable using only the operations in $\{+, *, \hat{}\}$.

The factorability can be defined for constraints as follows.

Definition 5 (Factorable Constraint)**.** *A constraint is said to be* factorable *in variables $x_1, \ldots, x_n$ using a finite set F of elementary operations if it can be expressed as a relation involving expressions that are factorable in variables $x_1, \ldots, x_n$ using operations in F. In the composition of a factorable constraint, each constraint representing an elementary operation is called a* primitive constraint.

In this paper, we restrict, for simplicity, the relation in a factorable constraint to be $\le$, $<$, $\ge$, $>$, $=$ or $\ne$. For example, the constraint $2x^y + \sin x \le 0$ is factorable in variables x and y using the operations in $\{+, *, \hat{}, \sin\}$. Its primitive constraints are $f_1 = x \hat{} y$ ($\equiv x^y$), $f_2 = 2 * f_1$, $f_3 = \sin(x)$, and $f_2 + f_3 \le 0$; where x and y are *initial variables*, and f_1, f_2 and f_3 are *auxiliary variables*.

The factorability can also be defined for a CSP as follows.

Definition 6. *A CSP is said to be* factorable *(using a set F of elementary operations) if all its constraints are factorable (using operations in F).*

2.2 Interval Arithmetic

Let $\mathbb{R}_\infty \equiv \mathbb{R} \cup \{-\infty, +\infty\}$. The *lower bound* of a real interval $\mathbf{x}$ is defined as $\inf(\mathbf{x})$, and the *upper bound* of $\mathbf{x}$ is defined as $\sup(\mathbf{x})$. Let denote $\underline{x} = \inf(\mathbf{x}) \in \mathbb{R}_\infty$ and $\overline{x} = \sup(\mathbf{x}) \in \mathbb{R}_\infty$. There are four possible intervals $\mathbf{x}$ with these bounds:

- The *closed interval* defined as $\mathbf{x} \equiv [\underline{x}, \overline{x}] \equiv \{x \in \mathbb{R} \mid \underline{x} \le x \le \overline{x}\}$;
- The *open interval* defined as $\mathbf{x} \equiv (\underline{x}, \overline{x}) \equiv \{x \in \mathbb{R} \mid \underline{x} < x < \overline{x}\}$;
- The *left-open interval* defined as $\mathbf{x} \equiv (\underline{x}, \overline{x}] \equiv \{x \in \mathbb{R} \mid \underline{x} < x \le \overline{x}\}$;
- The *right-open interval* defined as $\mathbf{x} \equiv [\underline{x}, \overline{x}) \equiv \{x \in \mathbb{R} \mid \underline{x} \le x < \overline{x}\}$.

The set of all closed intervals is denoted by $\mathbb{I}$ and the set of all intervals is denoted by $\mathbb{IR}$. The *interval hull* of a subset S of $\mathbb{R}$, denoted by $\square S$, is the smallest interval (w.r.t. the set inclusion) that contains S. For example, $\square((1,2] \cup \{3,4\}) = (1,4]$. Given a nonempty interval $\mathbf{x}$, we define that

- The *midpoint* of $\mathbf{x}$ is $\quad \mathrm{mid}(\mathbf{x}) \equiv (\inf(\mathbf{x}) + \sup(\mathbf{x}))/2;$
- The *radius* of $\mathbf{x}$ is $\quad \mathrm{rad}(\mathbf{x}) \equiv (\sup(\mathbf{x}) - \inf(\mathbf{x}))/2;$
- The *width* of $\mathbf{x}$ is $\quad \mathrm{w}(\mathbf{x}) \equiv \sup(\mathbf{x}) - \inf(\mathbf{x}).$

Note that the w, rad, and mid of the empty interval are undefined, as is mid for all unbounded intervals. A *box* is defined as the Cartesian product of a number of intervals. The concepts of the midpoint, radius and width can be defined on *boxes* in a component-wise manner. The set $\mathbb{IR}$ (hence the set $\mathbb{I}$) admits the usual *partial orders* $\diamond \in \{<, \leq, >, \geq\}$ as follows:

$$\mathbf{x} \diamond \mathbf{y} \quad \Leftrightarrow \quad \forall x \in \mathbf{x}, y \in \mathbf{y} \ : \ x \diamond y.$$

Interval arithmetic maintains the *inclusion property* that the result of an operation or function in interval arithmetic must enclose the exact range of its real-valued counterpart. For example, the addition of two real intervals, $\mathbf{x}$ and $\mathbf{y}$, can be defined as $\mathbf{x} + \mathbf{y} \equiv \{x + y \mid x \in \mathbf{x}, y \in \mathbf{y}\}$. Although the inclusion property characterizes the operations of interval arithmetic mathematically, the usefulness of interval arithmetic is due to the *operational definitions* based on interval bounds. For example, let $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$ be two intervals, the standard interval arithmetic shows that the natural extensions of the real operations can be computed as follows:

$$\mathbf{x} + \mathbf{y} \equiv [\underline{x} + \underline{y}, \overline{x} + \overline{y}]; \tag{1a}$$
$$\mathbf{x} - \mathbf{y} \equiv [\underline{x} - \overline{y}, \overline{x} - \underline{y}]; \tag{1b}$$
$$\mathbf{x} * \mathbf{y} \equiv [\min\{\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}\}, \max\{\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}\}]; \tag{1c}$$
$$\mathbf{x} \div \mathbf{y} \equiv \mathbf{x} * (1/\mathbf{y}) \quad \text{if } 0 \notin \mathbf{y}, \text{ where } 1/\mathbf{y} \equiv [1/\overline{y}, 1/\underline{y}]. \tag{1d}$$

One can compose arithmetic expressions with interval variables using these elementary operations in the way arithmetic expressions with real variables are composed. In general, one may wish to construct an *interval form*, $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}^m$, of a real function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ conforming to the inclusion property. That is, for all $\mathbf{x} \in \mathbb{IR}^n$, we have $\forall x \in \mathbf{x} \cap D : f(x) \in \mathbf{f}(\mathbf{x})$ or, for short, $f(\mathbf{x} \cap D) \subseteq \mathbf{f}(\mathbf{x})$.

Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a factorable function with one of its realizations $\mathfrak{f}$ as arithmetic expression. The *natural extension* of $\mathfrak{f}$ is an interval function $\mathbf{f}_{\mathfrak{f}} : \mathbb{IR}^n \to \mathbb{IR}^m$ constructed from $\mathfrak{f}$ in which each real variable is replaced by an interval variable and each elementary real operation is replaced by the natural extension of this operation. It is easy to prove that $\mathbf{f}_{\mathfrak{f}}$ is an interval form of the function f. Therefore, it is called a *natural interval form* of f and denoted by $\mathbf{f}$. Note that $\mathbf{f}$ indeed depends on the arithmetic expression used to realize f (see e.g. [Neumaier 1990, Chapter 1]), however, it is common usage to call every natural extension of f *the* natural (interval) extension of f. For example, the natural interval form of the real function $f(x, y) = 2{*}x{+}x{*}y$ is the interval function $\mathbf{f}(\mathbf{x}, \mathbf{y}) = 2{*}\mathbf{x}{+}\mathbf{x}{*}\mathbf{y}$. However, the same function f can be (better) represented by the interval function $\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{x}(2{+}\mathbf{y})$.

Rigorous enclosures The finite nature of number systems in computers precludes an exact representation of the real numbers. In practice, the real set $\mathbb{R}$ is hence approximated by a finite set $\mathbb{F}$, the set of *floating-point numbers* [Goldberg 1991], including the infinities. The set of real intervals is then replaced with the set $\mathbb{I}_\diamond$ of closed *floating-point intervals* with bounds in $\mathbb{F}$. The interval concepts are similarly defined on $\mathbb{I}_\diamond$ such that the inclusion property still holds. In addition, an interval in $\mathbb{I}_\diamond$ (respectively, a box in $\mathbb{I}_\diamond^n$) is said to be *canonical* if and only if it does not contain two disconnected intervals in $\mathbb{I}_\diamond$ (respectively, two disconnected boxes in $\mathbb{I}_\diamond^n$).[2] Usually, operations on floating-point intervals are outwardly rounded such that the floating-point result encloses the exact result. This interval arithmetic, called *outwardly rounded*, allows us to compute *rigorous enclosures* for the ranges of functions by using floating-point number systems.

The reader can find extended introductions to interval analysis in [Moore 1966, 1979; Alefeld and Herzberger 1983], interval methods for systems of equations in [Neumaier 1990], interval methods for optimization in [Hansen and Walster 2004], and some recent applications of interval arithmetic in [Jaulin et al. 2001].

2.3 Numerical Constraint Satisfaction

2.3.1 Numerical Constraint Satisfaction Problems

A *constraint* on a finite sequence of variables $(x_1, \dots, x_k)$ taking values in respective domains $(D_1, \dots, D_k)$ is a subset of the Cartesian product $D_1 \times \cdots \times D_k$, where k is a natural number, that is, in $\mathbb{N}$. The concept of a constraint satisfaction problem is defined as follows.

Definition 7. *A* constraint satisfaction problem*, abbreviated to CSP, is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ in which $\mathcal{V}$ is a finite sequence of variables $(v_1, \dots, v_n)$, $\mathcal{D}$ is a finite sequence of the respective domains of the variables, and $\mathcal{C}$ is a finite set of constraints (each on a subsequence of $\mathcal{V}$). A* solution *of this problem is an assignment of values from $\mathcal{D}$ to $\mathcal{V}$ respectively such that all constraints in $\mathcal{C}$ are satisfied. The set of all solutions is called the* solution set.

In this paper, we only focus on numerical CSPs defined as follows.

Definition 8. *A* numerical constraint *is a constraint on a sequence of variables whose domains are continuous, where a domain is called a* continuous domain *if it is a real interval. If all the constraints of a CSP are numerical, this CSP is called a* numerical constraint satisfaction problem *(abbreviated to NCSP).*

In practice, a NCSP can often be represented in the following form:

$$f(x) \in \mathbf{b}, \tag{2}$$

where x is a vector of n real variables taking values in a box $\mathbf{x} \in \mathbb{I}^n$, $\mathbf{b} \equiv (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m)^{\mathrm{T}}$ is an interval vector in $\mathbb{I}^m$, and $f = (f_1, f_2, \dots, f_m)^{\mathrm{T}}$ is a factorable function from $D \subseteq \mathbb{R}^n$ to $\mathbb{R}^m$. For each $j = 1, \dots, n$, the interval $\mathbf{b}_j$ is called the *constraint range* of the constraint $f_j(x) \in \mathbf{b}_j$.

Since more than thirty years ago, *constraint satisfaction* techniques, such as *arc consistency* [Waltz 1972, 1975; Montanari 1974] and *path consistency* Montanari [1974], have

2. Two intervals (or boxes) are disconnected if and only if their union is not an interval (a box, respectively).

been devised to solve CSPs with discrete domains. Those techniques perform *reasoning* procedures on constraints and explore the search space by intelligently enumerating solutions. In order to solve NCSPs by means of constraint satisfaction, continuous domains have often been converted into discrete domains by using progressive *discretization* techniques [Sam-Haroud 1995; Lottaz 2000]. Later on, many mathematical computation techniques for continuous domains have been integrated into the framework of constraint satisfaction in order to solve NCSPs more efficiently. Nowadays, these techniques are often referred to as *constraint programming*, which implies the combination of *computing* and *reasoning* aspects.

Most techniques for solving NCSPs follow the *branch-and-prune* framework, which interleaves branching steps with pruning steps. A *branching* step divides a problem into subproblems whose union is equivalent to the initial problem in term of the solution set, and a *pruning* step reduces a problem. Pruning steps are usually performed by using *domain reduction* techniques, which reduce the domains of variables without discarding any solution of the problem. Inspired by the classical constraint satisfaction techniques, an interesting approach in constraint programming, called *interval constraint propagation*, was developed in 1990s (see [Benhamou and Older 1992, 1997], [Van Hentenryck 1997] and [Jaulin et al. 2001]). This approach combines constraint propagation techniques in constraint satisfaction with interval-analytic methods in mathematics. The idea is that one cannot exactly achieve consistency properties such as arc consistency for numerical constraints under floating-point number systems, therefore replaces the consistencies with relaxations that are tractable under floating-point number systems. For example, given a constraint c on variables $(x_1, \ldots, x_k)$ with respective domains $(D_1, \ldots, D_k)$, arc consistency reduces each D_i to the projection of c on x_i, denoted $c[x_i]$. The interval variant of arc consistency only reduces each D_i to the smallest union of intervals that contains $c[x_i]$. However, this is still intractable in practice. One may wish to replace it with a weaker property that each D_i is the smallest interval containing $c[x_i]$. This introduces a new concept of consistency: *hull consistency* [Benhamou and Older 1992, 1997]. Other concepts of consistency such as kB-consistency [Lhomme 1993] and box consistency [Benhamou et al. 1994] have also been introduced. In constraint programming, achieving those consistency properties has often been implemented by a search or interval constraint propagation technique in combination with mathematical tools such as interval arithmetic.

2.3.2 Achieving Hull Consistency by Constraint Propagation

Let be given a factorable numerical constraint and one of its compositions. Benhamou and Older [1992, 1997] have proposed to achieve hull consistency for the initial constraint by achieving that for the primitive constraints of the initial constraint in the given composition (Definition 5). Benhamou et al. [1999] proposed a faster propagation algorithm to achieve hull consistency for a single constraint. To reduce the domains of the variables of a number of constraints, the technique, called **HC4**, achieves hull consistency for individual constraints, and then propagates the reduction of the variables' domains from constraint to constraint by using a variant of *arc consistency*, **AC3** [Mackworth 1977].

To solve a NCSP of the form (2), the **HC4** algorithm represents each constraint of the problem as a tree that defines a way to compose the constraint, where each node of the tree represents a primitive constraint (Definition 5). Each node $\mathbf{N}$ of the tree is associated with two intervals, called the *forward* and *backward node ranges*, denoted $\mathbf{N}^{\mathrm{f}}$ and $\mathbf{N}^{\mathrm{b}}$, respectively. The exact value, hence the exact range, of the subexpression represented by a node must be contained in both the node ranges.

Example 9. *The tree representation of the following NCSP is depicted in Figure 1:* $\{\sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \le 7,\ 0 \le x^2\sqrt{y} - 2xy + 3\sqrt{y} \le 2,\ x \in [1,16], y \in [1,16]\}$.

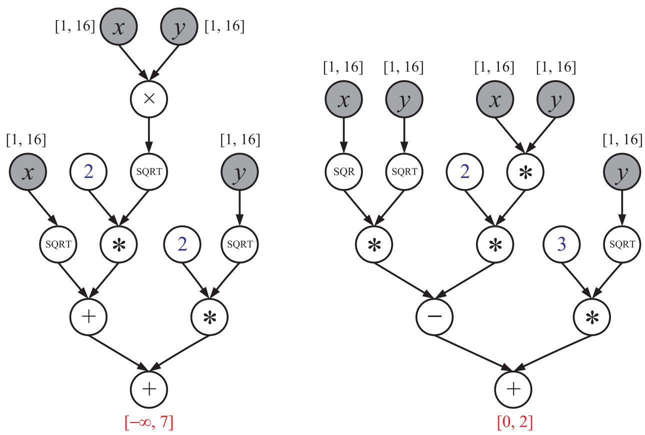

Figure 1: The tree representation of the NCSP in Example 9

The **HC4** algorithm is presented as Algorithm 1. It invokes another algorithm, **HC4revise**, to achieve hull consistency for a constraint. **HC4revise** performs two main processes: recursive forward evaluation (**RFE**) and recursive backward projection (**RBP**). **HC4revise** is presented concisely in Algorithm 2, where $\mathcal{T}_{\mathbf{N}}$ denotes the tree rooted at node $\mathbf{N}$. At Line 1 of **RBP**, an elementary operation $\psi(\mathbf{N}_1, \ldots, \mathbf{N}_q)$ represented by node $\mathbf{N}$ defines a relation ψ^* on the sequence $(\mathbf{N}, \mathbf{N}_1, \ldots, \mathbf{N}_q)$; where $\mathbf{N}, \mathbf{N}_1, \ldots, \mathbf{N}_q$ play the role of variables taking values in $\mathbf{N}^{\mathrm{b}}, \mathbf{N}_1^{\mathrm{f}}, \ldots, \mathbf{N}_q^{\mathrm{f}}$, respectively. Since ψ is an elementary operation, ψ^* is very simple and can be projected on its variables by using simple formulas in [Benhamou et al. 1999].

Algorithm 1: The **HC4** algorithm – hull consistency on primitive constraints

Input: a NCSP $\mathcal{P} \equiv (\mathcal{V} \equiv (x_1, \ldots, x_n), \mathcal{D}, \mathcal{C})$, a domain box $\mathbf{x} \subseteq \mathcal{D}$.
Output: new domains $\mathbf{x}' \in \mathbb{I}^n$ of $\mathcal{V}$.
$\mathbf{x}' := \mathbf{x}$; WaitingList $:= \mathcal{C}$;
while WaitingList $\neq \emptyset$ **and** $\mathbf{x}' \neq \emptyset$ **do**
    Take a constraint C from WaitingList;
1   $\mathbf{y} :=$ **HC4revise**$(\mathcal{T}_C, \mathbf{x}')$; ◀ On page 9.
    if $\mathbf{y} \neq \mathbf{x}'$ **then**
        Put into WaitingList the constraint C and every constraint C' sharing with C at least one variable whose domain has been reduced at Line 1;
        $\mathbf{x}' := \mathbf{y}$;

Algorithm 2: The **HC4revise** algorithm

Input: a tree $\mathcal{T}_C$; domains $\mathbf{x} \in \mathbb{I}^n$ of variables $(x_1, \ldots, x_n)$.
Output: new domains $\mathbf{x} \in \mathbb{I}^n$ of $(x_1, \ldots, x_n)$.
RFE$(\mathcal{T}_C, \mathbf{x})$; ◀ On page 9.
C^{b} := the constraint range of C;
RBP$(\mathcal{T}_C, \mathbf{x})$; ◀ On page 9.

Procedure RFE(in/out: a tree $\mathcal{T}_{\mathbf{N}}$; **in:** $\mathbf{x} \in \mathbb{I}^n$)

if $\mathbf{N}$ is a variable x_i **then** $\mathbf{N}^{\mathrm{f}} := \mathbf{x}_i$;
else if $\mathbf{N}$ is an expression $\psi(\mathbf{N}_1, \ldots, \mathbf{N}_q)$ **then**
  $\mathbf{N}^{\mathrm{f}} := \psi(\mathsf{RFE}(\mathcal{T}_{\mathbf{N}_1}, \mathbf{x}), \ldots, \mathsf{RFE}(\mathcal{T}_{\mathbf{N}_q}, \mathbf{x}))$;

Procedure RBP(in/out: a tree $\mathcal{T}_{\mathbf{N}}$, $\mathbf{x} \in \mathbb{I}^n$)

if $\mathbf{N}$ is a variable x_i **then** $\mathbf{x}_i := \mathbf{x}_i \cap \mathbf{N}^{\mathrm{b}}$;
else if $\mathbf{N}$ is an expression $\psi(\mathbf{N}_1, \ldots, \mathbf{N}_q)$ **then**
  $\mathbf{N}^{\mathrm{b}} := \mathbf{N}^{\mathrm{b}} \cap \mathbf{N}^{\mathrm{f}}$;
1 Let ψ^* be the relation $\mathbf{N} = \psi(\mathbf{N}_1, \ldots, \mathbf{N}_q)$ on interval vector $(\mathbf{N}^{\mathrm{b}}, \mathbf{N}_1^{\mathrm{f}}, \ldots, \mathbf{N}_q^{\mathrm{f}})^{\mathrm{T}}$;
  for $i := 1, \ldots, q$ **do**
    $\mathbf{N}_i^{\mathrm{b}} := \mathbf{N}_i^{\mathrm{f}} \cap \psi^*[\mathbf{N}_i]$; ◀ Intersected with the projection of ψ^* on $\mathbf{N}_i$.
    RBP$(\mathcal{T}_{\mathbf{N}_i}, \mathbf{x})$;

2.4 DAG Representations for Numerical CSPs

2.4.1 Directed Acyclic Graphs

For completeness, we recall hereafter some fundamental concepts in graph theory related to the concept of a DAG representation [Schichl and Neumaier 2005] of a constraint system.

Definition 10. *A* directed multigraph $\mathcal{G} \equiv (V, E, f)$ *consists of a finite set* V *of* vertices *(also called* nodes*), a finite set* E *of* edges *(also called* arcs*), and a mapping* $f \equiv (f_{\mathrm{s}}, f_{\mathrm{t}})^{\mathrm{T}} : E \to V \times V$ *such that for all* $e \in E$ *we have* $f_{\mathrm{s}}(e) \neq f_{\mathrm{t}}(e)$*. For every edge* $e \in E$*, we define the* source *of* e *as* $f_{\mathrm{s}}(e)$ *and the* target *of* e *as* $f_{\mathrm{t}}(e)$*.*

In the above definition, if we replace f with a function that maps each edge to an unordered pair of vertices, we then obtain the definition of a *multigraph.* In addition to that, if we allow the source and target of an edge be the same, then the obtained one is called a *pseudograph.* We can also obtain the concept of a *directed pseudograph* in the same way.

Definition 11. *Using the notations in Definition 10, we define the set of all* in-edges *of a vertex* $v \in V$ *as* $\mathsf{in\text{-}edges}(v) \equiv \{e \mid f_{\mathrm{t}}(e) = v\}$*. Similarly, we define the set of all* out-edges *of a vertex* $v \in V$ *as* $\mathsf{out\text{-}edges}(v) \equiv \{e \mid f_{\mathrm{s}}(e) = v\}$*.*

In other words, $\mathsf{in\text{-}edges}(v)$ is the set of all edges having v as their target and $\mathsf{out\text{-}edges}(v)$ is the set of all edges having v as their source. Similarly to a tree, the concepts of a leaf and a root in a directed multigraph are defined as follows.

Definition 12. *Consider a directed multigraph $\mathcal{G}$. A vertex v of $\mathcal{G}$ is called a* leaf *(or* local source*) of $\mathcal{G}$ if* $\mathsf{in\text{-}edges}(v) = \emptyset$. *A vertex v of $\mathcal{G}$ is called a* root *(or* local sink*) of $\mathcal{G}$ if* $\mathsf{out\text{-}edges}(v) = \emptyset$.

Unlike a (directed) tree, a directed multigraph may have many roots (and many leaves as well).

Definition 13. *Consider a directed multigraph $\mathcal{G} \equiv (V, E, f)$, where $f \equiv (f_s, f_t)^T$. A* directed path *from a vertex $v_1 \in V$ to a vertex $v_2 \in V$ is a sequence $(e_i)_{i=1}^n$ of edges, where $n \geq 2$, such that $v_1 = f_s(e_1)$, $v_2 = f_t(e_n)$, and $f_t(e_i) = f_s(e_{i+1})$ for all $i = 1, \ldots, n-1$. This directed path is called a* cycle *if $v_1 = v_2$. $\mathcal{G}$ is called a* directed acyclic multigraph *if it does not contain any cycle.*

Definition 14 (Parent, Child, Ancestor, Descendant)**.** *Consider a directed acyclic multigraph $\mathcal{G} \equiv (V, E, f)$. Let v_1 and v_2 be two vertices in V. We say that v_2 is a* parent *of v_1 and that v_1 is a* child *of v_2 if there exists an edge $e \in E$ such that $f(e) = (v_1, v_2)^T$. The set of all parents (respectively, all children) of a vertex $v \in V$ is denoted by* $\mathsf{parents}(v)$ *(respectively,* $\mathsf{children}(v)$*). We say that v_2 is an* ancestor *of v_1 and that v_1 is a* descendant *of v_2 if there exists a directed path from v_1 to v_2. The set of all ancestors (respectively, all descendants) of a vertex $v \in V$ is denoted by* $\mathsf{ancestors}(v)$ *(respectively,* $\mathsf{descendants}(v)$*).*

The following result is fundamental, which illustrates the precedence relationship of nodes in a directed acyclic multigraph.

Theorem 15. *For every directed acyclic multigraph (V, E, f), there exists a total order $\preceq$ on the vertices V such that for every $v \in V$ and every $u \in \mathsf{ancestors}(v)$, we have $v \preceq u$.*

Proof. See Procedure **NodeLevel** on page 23, which is is a simple algorithm for assigning a level to each node such that any sorting in descending order of the obtained levels will result in a required order. ■

Definition 16 (Directed Multigraph with Ordered Edges)**.** *A* directed multigraph with ordered edges *is a quadruple $(V, E, f, \leq)$ such that (V, E, f) is a directed multigraph and $(E, \leq)$ is a totally ordered set.*

2.4.2 DAG Representation

As proposed by Schichl and Neumaier [2005], a directed acyclic multigraph with ordered edges, abbreviated to *DAG*, can be used to represent the factorable NCSP (2). Since the problem (2) is factorable, the function f can be composed of a sequence of elementary operations/functions such as $+$, $*$, $/$, log, exp, sqr, and sqrt. In this composition, each variable is represented by a leaf. Each elementary operation/function $\phi : D \subseteq \mathbb{R}^k \to \mathbb{R}$ that takes as input k subexpressions $x_1, \ldots, x_k$ is represented by a node $\mathbf{N}$ with k edges, each runs from the node representing x_i to the node $\mathbf{N}$, where $1 \leq i \leq k$. These k edges represent the computational flow in the natural composition of the operation ϕ. The obtained representation is called the *DAG representation* of the considered problem.

Notation 17. *Each node $\mathbf{N}$ in the DAG representation is associated with an interval, denoted as $\tau_{\mathbf{N}}$ and called the* node range *of $\mathbf{N}$, in which the exact range of the associated*

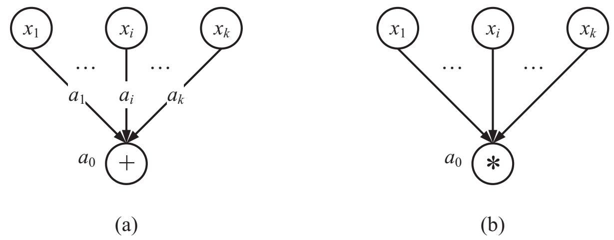

Figure 2: A node and its computational flows in a DAG representation.

subexpression must lie. **N** *is also associated with a real variable, denoted by* $\vartheta_{\mathbf{N}}$*, that represents the value of the subexpression represented by* **N**.

For efficiency and compactness, the standard elementary operations in the DAG representation are replaced with more general operations. For example, multiple applications of binary elementary operations of the forms in $\{x + y,\ x - y,\ x + a,\ a + x,\ x - a,\ a - x,\ ax\}$ are replaced with a k-ary operation $a_0 + a_1x_1 + \cdots + a_kx_k$, which is interpreted as a k-ary operation $+$ (see Figure 2a), where $1 \le k \in \mathbb{N}$. Similarly, multiple applications of the binary multiplication $x * y$ are replaced with a k-ary multiplication (or product) $a_0 * x_1 * \cdots * x_k$, which is interpreted as the k-ary operation $*$ (see Figure 2b), where $2 \le k \in \mathbb{N}$. In general, each edge of a DAG representation is associated with a respective coefficient of the operation represented by its target. When not specified in figures, this coefficient equals to 1. The other constants involving an operation are stored at the node representing the operation (see Figure 2). At a result, the DAG representation no longer have nodes representing constants as in the tree representation (see Section 2.3.2). Much more detailed descriptions of DAG representations can be found in Section 4.1 and Section 5.3 of [Schichl 2003].

We need to use multigraphs, for efficiency, instead of simple graphs for DAG representations because some special operations can take the same input more than once. For example, the expression x^x can be represented by the binary power operation x^y without introducing a new unary operation x^x. In all cases, a normal directed acyclic graph is sufficient to represent a numerical CSP (NCSP), provided that we introduce new elementary operations such as the unary operation x^x. The ordering of edges is needed for non-commutative operations like the division. For convenience, a *ground node*, called **G**, is added to each DAG representation to be the parent of all nodes that represents the constraints. In fact, the ground node can be interpreted as the logical `AND` operation.

Example 18. *Consider the following constraint system*

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \le 7, \\ 0 \le x^2\sqrt{y} - 2xy + 3\sqrt{y} \le 2, \\ x \in [1, 16],\ y \in [1, 16], \end{cases}$$

which can be written into the form (2) *as follows*

$$\begin{cases} \sqrt{x} + 2\sqrt{xy} + 2\sqrt{y} \in [-\infty, 7], \\ x^2\sqrt{y} - 2xy + 3\sqrt{y} \in [0, 2], \\ x \in [1, 16],\ y \in [1, 16]. \end{cases} \tag{3}$$

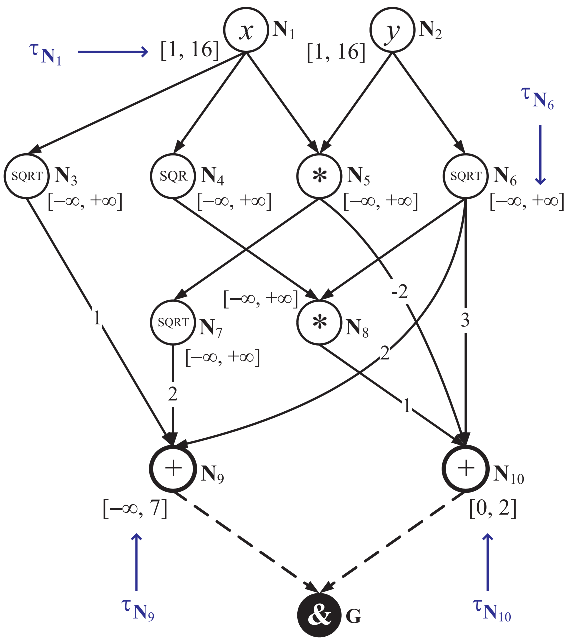

Figure 3: The DAG representation of the constraint system (3).

The DAG representation of the constraint system (3) *is depicted in Figure 3. Two constraints of* (3) *are represented by two nodes* $\mathbf{N}_9$ *and* $\mathbf{N}_{10}$*. Two variables, x and y, are represented by two nodes* $\mathbf{N}_1$ *and* $\mathbf{N}_2$*, respectively. The sequence* $(\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_{10})$ *of nodes given in Figure 3 is an example of an ordering as stated in Theorem 15.*

For the same constraint system, the DAG representation is clearly more concise than the tree representation described in Section 2.3.2.

3. Modification to Standard Interval Arithmetic

Expressions that are only defined on *subsets* of $\mathbb{R}^n$ are often encountered in practice. For example, a division by zero, such as $1 \div 0$, is not defined. Consequently, the division of two intervals is not defined in (standard) interval arithmetic when the denominator contains zero. In such cases, many implementations of interval arithmetic give, by convention, the universe interval $[-\infty, \infty]$ as result. This is an extension of (standard) interval arithmetic for all purposes, in order to conform to the inclusion property. If we use this implementation to evaluate the range of a function $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$, we will often get unnecessarily overestimated bounds of the form $[-\infty, \infty]$ in case the denominators of the divisions in f contain zero. In order to avoid such over-estimations, we have to extend functions depending on their use in specific computations. In this section, building on the concept of a *multifunction*, we propose a way to extend functions that are only defined on subsets of $\mathbb{R}^n$.

3.1 Extending Domains of Functions

We start by recalling the definition of a multifunction from [Singh et al. 1997, p. 34].

Definition 19 (Multifunction)**.** *Let X and Y be two sets. A* multifunction *F from X to Y (a* relation *on $X \times Y$), denoted as $F : X \to Y$, is a subset $F \subseteq X \times Y$. The inverse of F is a multifunction $F^{-1} : Y \to X$ defined by the rule: $(y, x) \in F^{-1} \Leftrightarrow (x, y) \in F$. We define the* values *of F at x to be $F(x) \equiv \{y \in Y \mid (x, y) \in F\}$, and the* fibers *of F for $y \in Y$ to be $F^{-1}(y) \equiv \{x \in X \mid (x, y) \in F\}$.*

In Definition 19, if for some $x \in X$ there is no $y \in Y$ such that $(x, y) \in F$, we have that $F(x) = \emptyset$. ¿From Definition 19 we can see that a function is, in fact, a special multifunction that is single-valued.

The concepts of image and inverse image (of a set) under a multifunction are similar to those for functions :

Definition 20 (Image, Inverse Image)**.** *Let X and Y be two sets, $F : X \to Y$ a multifunction. The* image of a subset *$A \subseteq X$* under *F is defined and denoted by*

$$F(A) \equiv \bigcup_{x \in A} F(x) = \{y \in F^{-1} \mid F^{-1}(y) \cap A \neq \emptyset\}. \tag{4}$$

The inverse image of a subset *$B \subseteq Y$* under *F is defined and denoted by*

$$F^{-1}(B) \equiv \bigcup_{y \in B} F^{-1}(y) = \{x \in F \mid F(x) \cap B \neq \emptyset\}. \tag{5}$$

Next, we define a special class of multifunctions.

Definition 21 (Extended Function)**.** *Let f be a function from a set X to a set Y, X' a superset of X, and Z a set of some subsets of Y possibly including $\emptyset$. A Z*-extended function over *X'* of *f is a multifunction $F : X' \to Y$ such that*

$$\forall x \in X \quad : \quad F(x) = \{f(x)\}, \tag{6}$$
$$\forall x \in X' \setminus X \quad : \quad F(x) \in Z. \tag{7}$$

Note 22. *When we do not care about Z in Definition 21, we just call F an extended function over X' of f.*

Notation 23. *For simplicity, in Definition 21, for all $x \in X$ we write $F(x) = f(x)$ when no confusion can arise.*

A Z-extended function F, as defined in Definition 21, corresponds to a function $g : X' \to Y \cup Z$ defined as

$$g(x) \equiv \begin{cases} f(x) & \text{if } x \in X, \\ F(x) & \text{otherwise.} \end{cases} \tag{8}$$

If $X' = X$, then $f(x) = g(x)$ for all x in X. By Definition 21, it is easy to prove the following theorem.

Theorem 24. *Let f, F and other notations be as in Definition 21. Then, for every subset S of X', we have*

$$f(S) \equiv \{f(x) \mid x \in S \cap X\} \subseteq F(S). \tag{9}$$

Consider the case $X = D \subseteq \mathbb{R}^n, Y = \mathbb{R}^m$. It is easy to see that, for any function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$, there is only one Z-extended function from $\mathbb{R}^n$ to $\mathbb{R}^m$ if Z has only one element, for example, when Z is either $\{\emptyset\}$ or $\{\mathbb{R}\}$.

Example 25. *The domain of the standard division $x \div y$ is $D_\div = \{(x, y) \in \mathbb{R}^2 \mid y \neq 0\}$. The unique $\{\emptyset\}$-extended function over $\mathbb{R}$ of the standard division is defined as*

$$\div_\emptyset(x, y) \equiv x \div_\emptyset y \equiv \begin{cases} x/y & \text{if } y \neq 0, \\ \emptyset & \text{otherwise.} \end{cases} \tag{10}$$

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

$$\div_\mathbb{R}(x, y) \equiv x \div_\mathbb{R} y \equiv \begin{cases} x/y & \text{if } y \neq 0, \\ \mathbb{R} & \text{otherwise.} \end{cases} \tag{11}$$

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

$$\div_\star(x, y) \equiv x \div_\star y \equiv \begin{cases} x/y & \text{if } y \neq 0, \\ \emptyset & \text{if } x \neq 0, y = 0, \\ \mathbb{R} & \text{otherwise.} \end{cases} \tag{12}$$

Example 26. *The domain of the standard square root $\sqrt{x}$ is the interval $[0, +\infty]$. The unique $\{\emptyset\}$-extended function over $\mathbb{R}$ of the square root is defined as*

$$\sqrt{x}^\emptyset \equiv \begin{cases} \sqrt{x} & \text{if } x \geq 0, \\ \emptyset & \text{otherwise.} \end{cases} \tag{13}$$

The unique $\{\mathbb{R}\}$-extended function over $\mathbb{R}$ of the square root is defined as

$$\sqrt{x}^\mathbb{R} \equiv \begin{cases} \sqrt{x} & \text{if } x \geq 0, \\ \mathbb{R} & \text{otherwise.} \end{cases} \tag{14}$$

3.2 Extending Interval Forms

We now define the concept of an *interval form* for a multifunction. This definition also holds for extended functions which are special cases of multifunctions.

Definition 27. *Let $F : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a multifunction. A function $[F] : \mathbb{I}^n \to \mathbb{I}^m$ is called an* interval form *of F if the following* inclusion property *holds:*

$$\forall x \in D, \forall \mathbf{x} \in \mathbb{I}^n \quad : \quad x \in \mathbf{x} \Rightarrow F(x) \subseteq [F](\mathbf{x}). \tag{15}$$

The *natural interval form* of f is an instance of an interval form. The following theorem states the inclusion property of interval forms of multifunctions.

Theorem 28. *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be a function and $F : D' \supseteq D \to \mathbb{R}^m$ an extended function over D' of f. Then every interval form of F is also an interval form of f.*

Proof. Let $[F] : \mathbb{I}^n \to \mathbb{I}^m$ be an interval form of F. Then for every $x \in D$ and every box $\mathbf{x} \in \mathbb{I}^n$ containing x, we have $f(x) \in \{f(x) \mid x \in \mathbf{x}\} = F(x) \subseteq [F](\mathbf{x})$. ∎

Definition 29 (Interval Division: $[\div_\emptyset]$, $[\div_\mathbb{R}]$, $[\div_\star]$)**.** *Let $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$ be two intervals. We define three natural interval forms of the division given by* (10)*,* (11) *and* (12)*, respectively:*

$$[\div_\emptyset](\mathbf{x},\mathbf{y}) \equiv \mathbf{x}[\div_\emptyset]\mathbf{y} \quad \equiv \quad \begin{cases} \emptyset & \text{if } \mathbf{y} = [0,0], \\ [0,0] & \text{else if } \mathbf{x} = [0,0], \\ \mathbf{x} \div \mathbf{y} & \text{else if } 0 \notin \mathbf{y} \text{ (see (1))}, \\ [\underline{x}/\overline{y}, +\infty] & \text{else if } \underline{x} \geq 0 \wedge \underline{y} = 0, \\ [-\infty, \underline{x}/\underline{y}] & \text{else if } \underline{x} \geq 0 \wedge \overline{y} = 0, \\ [-\infty, \overline{x}/\overline{y}] & \text{else if } \overline{x} \leq 0 \wedge \underline{y} = 0, \\ [\overline{x}/\underline{y}, +\infty] & \text{else if } \overline{x} \leq 0 \wedge \overline{y} = 0, \\ [-\infty, +\infty] & \text{otherwise;} \end{cases} \tag{16}$$

$$[\div_\mathbb{R}](\mathbf{x},\mathbf{y}) \equiv \mathbf{x}[\div_\mathbb{R}]\mathbf{y} \quad \equiv \quad \begin{cases} \mathbf{x} \div \mathbf{y} & \text{if } 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise;} \end{cases} \tag{17}$$

$$[\div_\star](\mathbf{x},\mathbf{y}) \equiv \mathbf{x}[\div_\star]\mathbf{y} \quad \equiv \quad \begin{cases} \mathbf{x}[\div_\emptyset]\mathbf{y} & \text{if } 0 \notin \mathbf{x} \vee 0 \notin \mathbf{y}, \\ [-\infty, +\infty] & \text{otherwise.} \end{cases} \tag{18}$$

Some authors [Hickey et al. 2001] use the tightest range of the division of two intervals; however, the result is not always an interval in that case.

Theorem 30. *For any two intervals $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{I}$, we have*

$$\mathbf{x}[\div_\emptyset]\mathbf{y} \quad \subseteq \quad \mathbf{x}[\div_\star]\mathbf{y} \quad \subseteq \quad \mathbf{x}[\div_\mathbb{R}]\mathbf{y}.$$

Proof. This is obvious from Definition 29. ∎

Theorem 31. *Let x, y, and z be three real numbers living in three intervals $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ in $\mathbb{I}$, respectively. Then we have*

$$x = y * z \quad \Rightarrow \quad z \in \mathbf{x} \diamond \mathbf{y} \quad \text{for all } \diamond \in \{[\div_\star], [\div_\mathbb{R}]\}, \tag{19}$$

$$z = x/y \quad \Rightarrow \quad z \in \mathbf{x} \diamond \mathbf{y} \quad \text{for all } \diamond \in \{[\div_\emptyset], [\div_\star], [\div_\mathbb{R}]\}. \tag{20}$$

Proof. Notice that from a given equality $x = y * z$ we can deduce the following:

- If $y \neq 0$, then $z = x/y$;
- If $y = 0$, then $x = 0$, and z can take an arbitrary value.

The rest of the proof follows directly from Definition 29. ∎

On one hand, Theorem 31 shows that, when given the relation $x = y * z$, it is safe to use the domain reduction $\mathbf{z} := \mathbf{x} \diamond \mathbf{y}$ for any $\diamond \in \{[\div_\star], [\div_\mathbb{R}]\}$. On the other hand, if the equality $z = x/y$ is given, we can safely reduce the domain of z by the rule $\mathbf{z} := \mathbf{x} \diamond \mathbf{y}$ for any $\diamond \in \{[\div_\mathbb{R}], [\div_\star], [\div_\emptyset]\}$, because the case $y = 0$ is not admitted by definition.

Theorems 30 and 31 show that the tightness of a natural interval form of a function defined on a subset of $\mathbb{R}^n$ usually dependens on the underlying extended function. In turn, the extended function should be chosen based on the context of the computation. Many interval implementations (e.g., [Walster et al. 2000]) use the division $[\div_{\mathbb{R}}]$ in all computations. However, from Theorem 31 we can see that it is safe to use the division $[\div_{\emptyset}]$ in computations such as forward evaluations and use the division $[\div_{\star}]$ in computations such as backward propagations, as described in Section 2.3.2 and Section 4.

4. Forward-Backward Propagation on DAG Representations

Schichl and Neumaier [2005] have adapted to DAGs the *forward evaluations* and *backward propagations* defined on trees [Benhamou et al. 1999]. The forward and backward procedures they propose work at the *graph level*, which means that all the nodes of the graph are forward evaluated then backward propagated at once.

In this section we shift the original definitions to the *node level*. The goal is to make it possible to run forward evaluation and backward propagation adaptively on particular nodes only. As shown in Section 5, the nodes will be chosen depending on their ability to cause changes in the domain ranges of their related expressions. This section also introduces the notion of a *partial DAG representation* which will make it possible to perform branch-and-prune search without creating multiple DAGs. This section uses Notation 17.

4.1 Forward Evaluation on DAG Representations

Forward evaluation at a node $\mathbf{N}$ is concerned with evaluating the range of the expression represented by $\mathbf{N}$ on the basis of the node ranges of the children of $\mathbf{N}$.

Consider the DAG representation of a factorable NCSP of the form (2). Let $\mathbf{N}$ be a node that is not the ground node and that has k children: $\mathbf{C}_1, \ldots, \mathbf{C}_k$. Suppose the operation represented by $\mathbf{N}$ is a function $h : D_h \subseteq \mathbb{R}^k \to \mathbb{R}$. The relation between $\mathbf{N}$ and its children is given by $\vartheta_{\mathbf{N}} = h(\vartheta_{\mathbf{C}_1}, \ldots, \vartheta_{\mathbf{C}_k})$. We define the forward evaluation at node $\mathbf{N}$ as follows.

Definition 32 (Forward Evaluation)**.** *Consider a node* $\mathbf{N}$ *and its operation* h *as described above. Let* $[h]$ *be an interval form of the* $\{\emptyset\}$*-extended function over* $\mathbb{R}$ *of* h*. The* forward evaluation at $\mathbf{N}$ using $[h]$ *is defined and denoted by*

$$\mathtt{FE}(\mathbf{N}, [h]) \quad \equiv \quad (\tau_{\mathbf{N}} := \tau_{\mathbf{N}} \cap [h](\tau_{\mathbf{C}_1}, \ldots, \tau_{\mathbf{C}_k})). \tag{21}$$

Example 33. *Consider the node* $\mathbf{N}_7$ *in Figure 3,* $h(z) \equiv \sqrt{z}$*, where* $z \equiv \vartheta_{\mathbf{N}_5}$*. We can use any interval form* $[h]$ *of the* $\{\emptyset\}$*-extended function over* $\mathbb{R}$ *of* h*, which is the function* $\sqrt{z}^{\emptyset}$ *defined by* (13)*, for the forward evaluation in* (21)*. We can use, for example, the natural interval form* $\mathbf{h}(\mathbf{z}) \equiv \sqrt{\mathbf{z}}$ *in place of* $[h]$ *in* (21)*.*

Remark 34. *We can also replace* $[h]$ *in* (21) *with an interval form of the recursive subexpression whose variables are the initial variables. For instance, we can replace the interval form* $[h]$ *of the node* $\mathbf{N}_7$ *in Figure 3 with the natural interval form of the recursive subexpression* ($\sqrt{xy}$) *composed of the nodes* $\mathbf{N}_7$*,* $\mathbf{N}_4$*,* $\mathbf{N}_1$*, and* $\mathbf{N}_2$*. That is, we can replace* $[h]$ *with the bivariate interval function* $\sqrt{\mathbf{xy}}$*.*

In our implementation, we use the natural interval form for simplicity. The natural interval form of the function $h(x_1, \ldots, x_k) = a_0 + a_1x_1 + \cdots + a_kx_k$ is the function

$\mathbf{h}(\mathbf{x}_1, \dots, \mathbf{x}_k) = a_0 + a_1\mathbf{x}_1 + \cdots + a_k\mathbf{x}_k$.[3] Similarly, the natural interval form of the function $h(x_1, \dots, x_k) = ax_1 \dots x_k$ is the function $\mathbf{h}(\mathbf{x}_1, \dots, \mathbf{x}_k) = a\mathbf{x}_1 \dots \mathbf{x}_k$. The division of two reals has multiple natural interval forms because it is not defined when the denominator is zero (see Section 3.1 and Section 3.2). In Definition 29, we have provided three versions that can be called the natural interval forms of the real division: $[\div_\emptyset]$, $[\div_\star]$, and $[\div_\mathbb{R}]$. They all can be used in the forward evaluation defined by (21) if h is the real division.

Theorem 35 (Correctness)**.** *Consider the DAG representation of a factorable numerical CSP given in* (2)*. The forward evaluation defined in Definition 32, when applied to any node, never discards a solution of the considered problem.*

Proof. For every solution of the considered problem, there exists an assignment of values from the intervals $\tau_\mathbf{N}$, $\tau_{\mathbf{C}_1}, \dots, \tau_{\mathbf{C}_k}$ to the variables $\vartheta_\mathbf{N}$, $\vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_k}$, respectively, such that $\vartheta_\mathbf{N} = h(\vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_k})$. Because $[h]$ is an interval form of the $\{\emptyset\}$-extended function over $\mathbb{R}$ of h, it follows from Theorem 28 that $h(\vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_k}) \in [h](\tau_{\mathbf{C}_1}, \dots, \tau_{\mathbf{C}_k})$. Thus, $\vartheta_\mathbf{N} \in \tau_\mathbf{N} \cap [h](\tau_{\mathbf{C}_1}, \dots, \tau_{\mathbf{C}_k})$. The proof is, therefore, complete. ∎

4.2 Backward Propagation on DAG Representations

Backward propagation at a node $\mathbf{N}$ will reduce the node range of each child of $\mathbf{N}$ on the basis of the node ranges of $\mathbf{N}$ and on the node ranges of its other children.

Consider the DAG representation of a factorable NCSP of the form (2). Let $\mathbf{N}$ be a node that is not the ground node and that has k children: $\mathbf{C}_1, \dots, \mathbf{C}_k$. The operation represented by $\mathbf{N}$ is a function $h : D_h \subseteq \mathbb{R}^k \to \mathbb{R}$. The *backward propagation* attempts to prune each node range $\tau_{\mathbf{C}_i}$ of $\mathbf{C}_i$ based on the node range $\tau_\mathbf{N}$ of $\mathbf{N}$ and based on the node ranges of the other children, where $1 \le i \le k$. In other words, for each child $\mathbf{C}_i$, the backward propagation attempts to encloses the i-th projection of the relation $\vartheta_\mathbf{N} = h(\vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_k})$ on the variable $\vartheta_{\mathbf{C}_i}$ in a tight interval. This procedure is called the i-th backward propagation at $\mathbf{N}$ and denoted by $\mathtt{BP}(\mathbf{N}, \mathbf{C}_i)$. We define the following as the *backward propagation at* $\mathbf{N}$:

$$\mathtt{BP}(\mathbf{N}) \equiv \{\mathtt{BP}(\mathbf{N}, \mathbf{C}_1), \dots, \mathtt{BP}(\mathbf{N}, \mathbf{C}_k)\}. \tag{22}$$

Although the exact projection of a relation is expensive, in general, an enclosure of the exact projection of an elementary operation can often be obtained at low cost. Indeed, suppose that we can infer from the relation $\vartheta_\mathbf{N} = h(\vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_k})$ an equivalent relation

$$\vartheta_{\mathbf{C}_i} = g_i(\vartheta_\mathbf{N}, \vartheta_{\mathbf{C}_1}, \dots, \vartheta_{\mathbf{C}_{i-1}}, \vartheta_{\mathbf{C}_{i+1}}, \dots, \vartheta_{\mathbf{C}_k})$$

for some $i \in \{1, \dots, k\}$, where g_i is a function from $D_g \subseteq \mathbb{R}^k$ to $\mathbb{R}$ such that

$$D_g \supseteq \tau_\mathbf{N} \times \tau_{\mathbf{C}_1} \times \cdots \times \tau_{\mathbf{C}_{i-1}} \times \tau_{\mathbf{C}_{i+1}} \times \cdots \times \tau_{\mathbf{C}_k}.$$

Let $[g_i]$ be an interval form of the $\{\emptyset\}$-extended function over $\mathbb{R}$ of g_i. The i-th backward propagation, denoted $\mathtt{BP}(\mathbf{N}, \mathbf{C}_i)$, can then be defined as

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \equiv (\tau_{\mathbf{C}_i} := \tau_{\mathbf{C}_i} \cap [g_i](\tau_\mathbf{N}, \tau_{\mathbf{C}_1}, \dots, \tau_{\mathbf{C}_{i-1}}, \tau_{\mathbf{C}_{i+1}}, \dots, \tau_{\mathbf{C}_k})). \tag{23}$$

3. Note that if the coefficients $a_0, \dots, a_k$ are real and we are working on the floating-point number system, we can replace each a_i in $\mathbf{h}$ with the smallest interval containing it, where $1 \le i \le k$.

In case we cannot infer such a function g_i, more complicated rules have to be constructed in order to obtain the i-th projection of the relation $\mathbf{N} = f(\mathbf{C}_1, \dots, \mathbf{C}_k)$ if the cost is low, otherwise the relation can be ignored. Fortunately, we can tightly enclose such projections at low cost for most elementary operations, as shown in Definition 38.

Remark 36. *In general, the relation $x = y * z$ and the relation $z = x/y$ are not equivalent because the latter discards the case $y = 0$ while the former does not.*

Example 37. *Consider the node $\mathbf{N}_{10}$ in Figure 3. The relation given at $\mathbf{N}_{10}$ is $\vartheta_{\mathbf{N}_{10}} = h(\vartheta_{\mathbf{N}_5}, \vartheta_{\mathbf{N}_6}, \vartheta_{\mathbf{N}_8})$, where the function h is defined as $h(x_1, x_2, x_3) \equiv -2x_1 + 3x_2 + x_3$. Therefore, we can infer three equivalent relations:*

$$\begin{aligned}
\vartheta_{\mathbf{N}_5} &= g_1(\vartheta_{\mathbf{N}_{10}}, \vartheta_{\mathbf{N}_6}, \vartheta_{\mathbf{N}_8}),\\
\vartheta_{\mathbf{N}_6} &= g_2(\vartheta_{\mathbf{N}_{10}}, \vartheta_{\mathbf{N}_5}, \vartheta_{\mathbf{N}_8}),\\
\vartheta_{\mathbf{N}_8} &= g_3(\vartheta_{\mathbf{N}_{10}}, \vartheta_{\mathbf{N}_5}, \vartheta_{\mathbf{N}_6}),
\end{aligned}$$

where the three functions g_1, g_2 and g_3 are defined as follows:

$$\begin{aligned}
g_1(x_1, x_2, x_3) &\equiv (-x_1 + 3x_2 + x_3)/2,\\
g_2(x_1, x_2, x_3) &\equiv (x_1 + 2x_2 - x_3)/3,\\
g_3(x_1, x_2, x_3) &\equiv x_1 + 2x_2 - 3x_3.
\end{aligned}$$

Definition 38 (Backward Propagation Rule)**.** *Let h be the elementary operation represented by node $\mathbf{N}$, as discussed above. We 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 backward propagation are given as follows:*

1. *If h is a univariate function such as* sqr, sqrt, exp, *and* log *and if $[h]$ is an interval form of the $\{\emptyset\}$-extended function of h, we define*

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_1) \quad \equiv \quad \left(\tau_{\mathbf{C}_1} := \tau_{\mathbf{C}_1} \cap [h^{-1}](\tau_\mathbf{N})\right),$$

   where the notation of interval form, $[h^{-1}](\mathbf{x})$, shall denote the union of some intervals that contains the inverse image $h^{-1}(\mathbf{x})$;

2. *If h is defined as $h(x_1, \dots, x_k) \equiv a_0 + a_1 x_1 + \cdots + a_k x_k$, we define for $i = 1, \dots, k$:*

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \quad \equiv \quad \left(\tau_{\mathbf{C}_i} := \tau_{\mathbf{C}_i} \cap \left((\tau_\mathbf{N} - a_0 \; - \sum_{j=1;\; j \neq i}^{k} a_j * \tau_{\mathbf{C}_j}) \oslash a_i\right)\right);$$

3. *If h is defined as $h(x_1, \dots, x_k) \equiv a x_1 \dots x_k$, we define for $i = 1, \dots, k$:*

$$\mathtt{BP}(\mathbf{N}, \mathbf{C}_i) \quad \equiv \quad \left(\tau_{\mathbf{C}_i} := \tau_{\mathbf{C}_i} \cap \left(\tau_\mathbf{N} \oslash (a \; * \prod_{j=1;\; j \neq i}^{k} \tau_{\mathbf{C}_j})\right)\right);$$

4. If h is defined as $h(x, y) \equiv x/y$, we define

$$\begin{aligned}
\texttt{BP}(\mathbf{N}, \mathbf{C}_1) &\equiv (\tau_{\mathbf{C}_1} := \tau_{\mathbf{C}_1} \cap (\tau_{\mathbf{N}} * \tau_{\mathbf{C}_2}))\,, \\
\texttt{BP}(\mathbf{N}, \mathbf{C}_2) &\equiv (\tau_{\mathbf{C}_2} := \tau_{\mathbf{C}_2} \cap (\tau_{\mathbf{C}_1} \oslash \tau_{\mathbf{N}}))\,.
\end{aligned}$$

The following theorem states the correctness of the backward propagation rules given in Definition 38.

Theorem 39 (Correctness)**.** *Consider the DAG representation of a factorable numerical CSP given in* (2)*. The backward propagation defined in Definition 38, when applied to any node, never discards any solution of the considered problem.*

Proof. By an argument similar to the proof of Theorem 35, we have that the first result is due to the definition of h^{-1} in Definition 20, and the other results are due to Theorem 31 and the inclusion property of the operations $+$, $-$, and $*$ in (standard) interval arithmetic. ■

4.3 Partial DAG Representations

When solving NCSPs using a *branch-and-prune* scheme, the *branching step* splits the problem into subproblems, potentially easier to solve. Each subproblem often consists of the following two components:

1. a subset of the initial constraints set, called the set of *running constraints*;

2. a sequence of subdomains for the involved variables.

If we use DAG representations in the pruning steps, we have to construct a DAG representation for each subproblem. A simple way is to construct a new DAG explicitly to represent each subproblem. However, the total cost of creating such DAGs for the whole solving process is potentially high, because there are often a huge number of branching steps during the solution process.

Alternatively, we propose to attach a piece of restriction information to the initial DAG, which is the DAG representation of the initial problem, in order to interpret the initial DAG as the DAG representation of a subproblem without having to create a new DAG. When using such pieces of restriction information, it is possible to perform forward evaluations and backward propagations on the DAG representation of the initial problem without increasing the time and space for dealing with DAG representations. A combination of such a piece of restriction information and the DAG representation of the initial problem is called the *partial DAG representation* of a subproblem. It is also called, for convenience, a partial DAG representation of the initial problem. For example, partial DAG representations of the problem (3) are depicted in Figure 4. We use partial DAG representations instead of DAG representations in our new propagation algorithm (Section 5).

In order to represent a subproblem with a set of running constraints without having to create a new DAG, we use a vector V_{oc} whose size equals the number of nodes in the DAG representation $D_{\mathbf{G}}$ of the initial problem. For each node $\mathbf{N}$ of $D_{\mathbf{G}}$, we use the entry $V_{\text{oc}}[\mathbf{N}]$ to count the number of occurrences of $\mathbf{N}$ in the recursive composition of the running

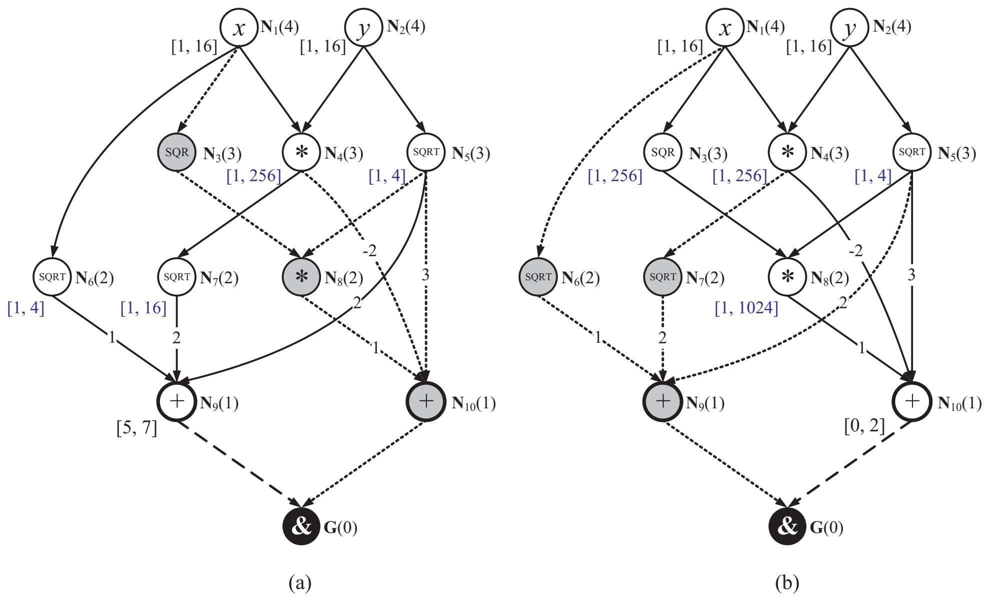

Figure 4: The partial DAG representations of the problem (3) in the cases: (a) the first constraint is the unique running constraint; and (b) the second constraint is the unique running constraint. The grey nodes and dotted edges are ignored. The node levels are given in parenthesis.

Procedure **NodeOccurrences**(**in:** a node $\mathbf{N}$; **in/out:** a vector V_{oc})

---

foreach $\mathbf{C} \in \mathsf{children}(\mathbf{N})$ **do**
  $V_{\mathrm{oc}}[\mathbf{C}] := V_{\mathrm{oc}}[\mathbf{C}] + 1;$
  NodeOccurrences($\mathbf{C}$, V_{oc});

---

constraints. We present a simple recursive procedure, called **NodeOccurrences**, to compute such a vector.

If we invoke **NodeOccurrences** at all the nodes representing the running constraints, then each entry $V_{\mathrm{oc}}[\mathbf{N}]$ will contains the number of occurrences of $\mathbf{N}$ in the recursive composition of the running constraints. In particular, we have $V_{\mathrm{oc}}[\mathbf{N}] = 0$ if and only if $\mathbf{N}$ is not in the representation of the running constraints. Therefore, by combining $D_{\mathbf{G}}$ with a vector V_{oc}, we have the so-called *partial DAG representation* for a subproblem. In computations, we can use partial DAG representations in a way similar to the way we use DAG representations, except that we ignore every node $\mathbf{N}$ correponding to $V_{\mathrm{oc}}[\mathbf{N}] = 0$.

5. Constraint Propagation on Partial DAG Representations

This section presents our new algorithm, called **FBPD** which generalizes to DAGs the **HC4** algorithm originally proposed by [Benhamou et al. 1999] for tree representations of constraints.

Algorithm 6: The **FBPD** algorithm – a constraint propagation on DAGs

Input: a DAG, $D_{\mathbf{G}}$, with the ground $\mathbf{G}$, domains $\mathcal{D}$, running constraints $\mathcal{C}$.
Output: new domains $\mathcal{D}$.
Reset all node ranges of $D_{\mathbf{G}}$ to $[-\infty, \infty]$;
Set the node ranges of variables & constraints to $\mathcal{D}$ & the constraint ranges of $\mathcal{C}$, resp.;
$\mathcal{L}_\mathrm{f} := \emptyset$; $\mathcal{L}_\mathrm{b} := \emptyset$; $V_\mathrm{oc} := (0, \ldots, 0)$; $V_\mathrm{ch} := (0, \ldots, 0)$;
1 $V_\mathrm{lvl} := (0, \ldots, 0)$; ◀ This can be made optional together with Line 2.
foreach node $\mathbf{C}$ representing a running constraint in $\mathcal{C}$ **do**
  NodeOccurrences($\mathbf{C}$, V_oc); ◀ On page 20.
2   **NodeLevel**($\mathbf{C}$, V_lvl); ◀ This can be made optional together with Line 1. On page 23.
3   **ReForwardEvaluation**($\mathbf{C}$, V_ch, $\mathcal{L}_\mathrm{b}$); ◀ A full recursive forward evaluation. On page 22.
  if the `infeasible` status was detected **then** **return** $\mathcal{D} := \emptyset$;
while $\mathcal{L}_\mathrm{b} \neq \emptyset \vee \mathcal{L}_\mathrm{f} \neq \emptyset$ **do**
4   $\mathbf{N} :=$ getNextNode($\mathcal{L}_\mathrm{b}, \mathcal{L}_\mathrm{f}$);
  if $\mathbf{N}$ was taken from $\mathcal{L}_\mathrm{b}$ **then**
    foreach child $\mathbf{C}$ of $\mathbf{N}$ **do**
5   BP($\mathbf{N}, \mathbf{C}$); ◀ See Definition 38.
      if $\tau_{\mathbf{C}} = \emptyset$ **then** **return** $\mathcal{D} := \emptyset$; ◀ The `infeasible` status was detected.
6   **if** the change of $\tau_{\mathbf{C}}$ is amenable to doing forward evaluations **then**
        foreach $\mathbf{P} \in$ parents($\mathbf{C}$) $\setminus \{\mathbf{N}, \mathbf{G}\}$ **do**
          if $V_\mathrm{oc}[\mathbf{P}] > 0$ **then** Put $\mathbf{P}$ into $\mathcal{L}_\mathrm{f}$; ◀ $\mathbf{P}$ occurs in a running constraint.
7   **if** the change of $\tau_{\mathbf{C}}$ is amenable to doing a backward propagation **then**
        Put $\mathbf{C}$ into $\mathcal{L}_\mathrm{b}$;
  else ◀ $\mathbf{N}$ was taken from $\mathcal{L}_\mathrm{f}$.
8   FE($\mathbf{N}, [h]$); ◀ h is the operator at $\mathbf{N}$, $[h]$ is an interval form of h, see Definition 32.
    if $\tau_{\mathbf{N}} = \emptyset$ **then** **return** $\mathcal{D} := \emptyset$; ◀ The `infeasible` status was detected.
9   **if** the change of $\tau_{\mathbf{N}}$ is amenable to doing forward evaluations **then**
      foreach $\mathbf{P} \in$ parents($\mathbf{N}$) $\setminus \{\mathbf{G}\}$ **do**
        if $V_\mathrm{oc}[\mathbf{P}] > 0$ **then** Put $\mathbf{P}$ into $\mathcal{L}_\mathrm{f}$; ◀ $\mathbf{P}$ occurs in some running constraint.
10   **if** the change of $\tau_{\mathbf{N}}$ is amenable to doing a backward propagation **then**
      Put $\mathbf{N}$ into $\mathcal{L}_\mathrm{b}$;
Update $\mathcal{D}$ with the node ranges of the variables;

We recall that, at each iteration, the **HC4** algorithm invokes the **HC4revise** algorithm (see Section 2.3.2), which in turn consists of two recursive propagation procedures: a recursive forward evaluation (**RFE**) and a recursive backward propagation (**RBP**). In order to reduce the node ranges and, in particular, the domains of variables, **RFE** performs forward evaluations at all nodes of the tree representation of a constraint in the post-order and then **RBP** performs backward propagations at all nodes of this tree representation in the pre-order.

Consider a factorable NCSP. We propose in this section a new propagation algorithm that enhances the **HC4** algorithm by:

- working on (partial) DAG representations, instead of tree representations, of the considered problem;

- exploiting the common subexpressions of the constraints as the influence of the constraints on each other;
- flexibly choosing nodes at which the forward evaluations and backward propagations are performed.

Moreover, the nature of the new propagation algorithm makes it possible to use different interval forms at different steps of the propagation. As discussed above, the new propagation algorithm works on partial DAG representations of the initial problem to reduce time and space when dealing with DAG representations. Since the main processes of the new algorithm are forward evaluations and backward propagations, we call it the *Forward-Backward Propagation on DAGs* (**FBPD**). The main steps of the **FBPD** algorithm are presented in Algorithm 6.

The **FBPD** algorithm takes as input a subproblem that is represented by the DAG representation $D_{\mathbf{G}}$ of the initial problem, a sequence $\mathcal{D}$ of subdomains of variables, and a set $\mathcal{C}$ of running constraints of the subproblem. Like the **HC4** algorithm, the **FBPD** algorithm relies on two types of processes: forward evaluation and backward propagation. Unlike the **HC4** algorithm, the **FBPD** algorithm, however, performs these processes on the basis of one node at a time rather than all nodes at once. The choice of the next node for the next process in the **FBPD** algorithm is adaptively made based on the results of the previous processes. Moreover, in the **FBPD** algorithm the choice of the interval form $[h]$ of an operation h for forward evaluations and backward propagations is not necessarily fixed. The interval form $[h]$ can be chosen statically or dynamically based on the nature of h at the current context.

In the next subsections, we describe in detail the procedures that are not made explicit in Algorithm 6.

5.1 Initialization Phase

Similarly to the **HC4** algorithm, the **FBPD** algorithm performs a recursive forward evaluation at the initialization phase (Line 3 in Algorithm 6) to evaluate the node ranges of all nodes in the partial DAG representation of the subproblem. That is, **FBPD** computes the node ranges of the nodes of $D_{\mathbf{G}}$ that correspond to nonzero entries in V_{oc}. Procedure **ReForwardEvaluation** provides such an algorithm. In order to avoid evaluating the same subexpressions multiple times, we use a vector, V_{ch}, to mark the caching status of

Procedure ReForwardEvaluation(**in**: a node $\mathbf{N}$; **in/out**: a vector V_{ch}, a list $\mathcal{L}_{\mathrm{b}}$)

```
if N is a leaf or V_ch[N] = 1 then return;        ◀ N is a leaf or has been cached.
foreach C ∈ children(N) do
    ReForwardEvaluation(C, V_ch, L_b);
if N = G then return;
FE(N, [h]);                                       ◀ This is similar to Line 8 in Algorithm 6.
V_ch[N] := 1;                                     ◀ The node range of N is cached.
if τ_N = ∅ then return infeasible;
if the change of τ_N is amenable to doing a backward propagation then
    Put C into L_b;
```

node ranges. A node $\mathbf{N}$ is marked as "cached" by setting $V_{\text{ch}}[\mathbf{N}] := 1$ if its node range has already been computed.

The result of the recursive forward evaluation of the NCSP given in (3) is depicted in Figure 4 (in case only one constraint is running in the subproblem) and Figure 5 (in case both constraints are running in the subproblem).

5.2 Getting the Next Node

The **FBPD** algorithm uses two waiting lists, $\mathcal{L}_\text{f}$ and $\mathcal{L}_\text{b}$, to store the nodes waiting for further processing. The first list, $\mathcal{L}_\text{f}$, is a list of nodes that is scheduled for forward evaluation, that is, for evaluating its node range based on the node ranges of its children. The second list, $\mathcal{L}_\text{b}$, is a list of nodes that is waiting for backward propagation, that is, for reducing the node ranges of its children based on its node range. In general, the nodes in $\mathcal{L}_\text{f}$ should be sorted such that the forward evaluation at a node is performed after the forward evaluations at its children. Analogously, the nodes in $\mathcal{L}_\text{b}$ should be sorted such that the backward propagation at a node is performed before the backward propagations at its children.

Procedure **NodeLevel** 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 (see Theorem 15). We then sort the nodes of $\mathcal{L}_\text{b}$ and $\mathcal{L}_\text{f}$ in ascending order and descending order of node levels, respectively, to meet the above requirements.

The call to Procedure **NodeLevel** at Line 2 in Algorithm 6 can be made optional as follows. The first option is to invoke **NodeLevel** only at the first call to the **FBPD** algorithm.

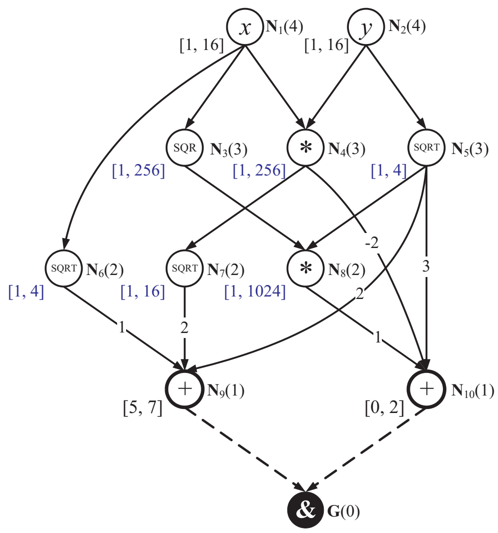

Figure 5: The DAG representation of the system (3) after a recursive forward evaluation.

Procedure NodeLevel(in: a node $\mathbf{N}$; **in/out:** a vector V_{lvl})

foreach child $\mathbf{C}$ of node $\mathbf{N}$ **do**
 $V_{\text{lvl}}[\mathbf{C}] := \max\{V_{\text{lvl}}[\mathbf{C}], V_{\text{lvl}}[\mathbf{N}] + 1\}$;
 NodeLevel($\mathbf{C}$, V_{lvl});

The node levels of the initial DAG still meet the requirements on the ordering of the waiting lists. The numbers in brackets following the node names in Figure 4 are the node levels computed for the initial DAG representation. Figure 6 illustrates the second option that is to invoke **NodeLevel** at Line 2 in Algorithm 6 each time the **FBPD** algorithm is invoked.

The getNextNode function at Line 4 in Algorithm 6 chooses one of the two nodes at the beginning of $\mathcal{L}_\text{b}$ and $\mathcal{L}_\text{f}$. The choosing strategy we use in our implementation is *backward propagation first*, that is, taking the node at the beginning of $\mathcal{L}_\text{b}$ whenever $\mathcal{L}_\text{b}$ is not empty. Of course, more involved strategies can also be considered.

5.3 Is the Change of a Node Range Amenable to Further Processing?

For simplicity, at the lines 6, 7, 9 and 10 of Algorithm 6 we only briefly present the procedures to check if the change of a node range is amenable to a forward evaluation or backward propagation. Hereafter, we describe them in detail.

Let $\mathbf{M}$ denote the node $\mathbf{C}$ at Line 5 or the node $\mathbf{N}$ at Line 8 in Algorithm 6. The backward propagation at Line 5 and the forward evaluation at Line 8 in Algorithm 6 have the same form

$$\tau_\mathbf{M} := \tau_\mathbf{M} \cap \mathbf{y}, \tag{24}$$

where $\mathbf{y}$ is the interval computed by the forward evaluation or backward propagation right before intersecting with $\tau_\mathbf{M}$ at the considered line. Let W_old and W_new be the widths of $\tau_\mathbf{M}$ and $\tau_\mathbf{M} \cap \mathbf{y}$, respectively, right before the intersection.

In practice, the change of $\tau_\mathbf{M}$ after performing (24) is amenable to doing forward evaluations at $\mathbf{M}$'s parents if both conditions $W_\text{new} < r_\text{f} * W_\text{old}$ and $W_\text{new} + d_\text{f} < W_\text{old}$ hold, where $r_\text{f} \in (0, 1]$ and $d_\text{f} \geq 0$ are real parameters.

Similarly, the change of $\tau_\mathbf{M}$ after performing the intersection (24) is amenable to doing a backward propagation at $\mathbf{M}$ if both conditions $W_\text{new} < r_\text{b} * W_\text{old}$ and $W_\text{new} + d_\text{b} < W_\text{old}$

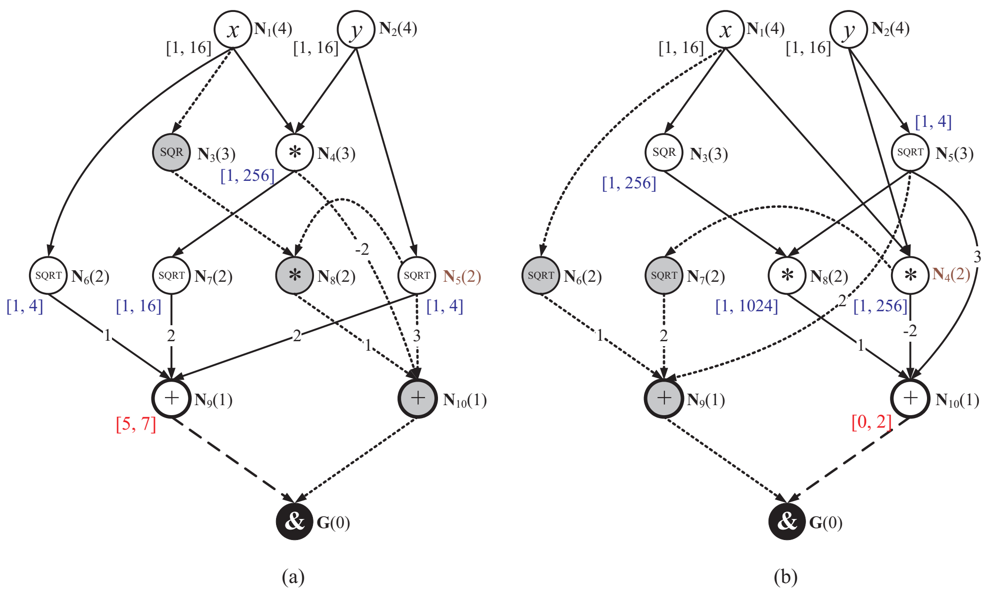

Figure 6: The node levels are updated at each call to the **FBPD** algorithm

hold, where $r_\mathrm{b} \in (0, 1]$ and $d_\mathrm{b} \geq 0$ are real parameters. In addition to that, the condition $\mathbf{y} \nsubseteq \tau_\mathbf{M}$ must also hold if $\mathbf{y}$ has been computed by the forward evaluation (at Line 8).

The parameters r_f, d_f, r_b and d_b can be predetermined or dynamically computed. In our implementation these parameters are predetermined.

5.4 Properties of the New Propagation Algorithm

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

Theorem 40. *Let $\Phi : \mathbb{I}^n \to \mathbb{I}^n$ be a function representing the* **FBPD** *algorithm. This function takes as input the domains of the input problem in the form of a box $\mathbf{x} \in \mathbb{I}^n$ and returns a box in $\mathbb{I}^n$, denoted as $\Phi(\mathbf{x})$, that represents the domains of the output problem of the* **FBPD** *algorithm. If the input problem contains only the operations h defined in Definition 32 and Definition 38, then the* **FBPD** *algorithm terminates at a finite number of iterations and the following properties hold:*

$$\text{(Contractiveness)} \qquad \Phi(\mathbf{x}) \subseteq \mathbf{x}, \tag{25}$$

$$\text{(Correctness)} \qquad \Phi(\mathbf{x}) \supseteq \mathbf{x} \cap S, \tag{26}$$

where S is the exact solution set of the input problem.

Proof. All the node ranges in the DAG representation of the considered problem are never inflated at each step of the **FBPD** algorithm. Hence, the **FBPD** algorithm must terminate at a finite number of iterations because of the finite nature of floating-point numbers. In particular, the ranges of the nodes representing the variables are never inflated. Thus, the property (25) holds. Moreover, the forward evaluations and backward propagations used in the **FBPD** algorithm are defined in Definition 32 and Definition 38. It follows from Theorem 35 and Theorem 39 that they never discard a solution. Therefore, the property (26) also holds. ■

Theorem 41. *Let $\mathbf{M}$ be a node in the (partial) DAG representation of the output subproblem of the* **FBPD** *algorithm.*

- *Suppose that the* **FBPD** *algorithm uses a fixed interval form $[h]$ of the elementary operation h presented by $\mathbf{M}$ at all steps. Let $\mathbf{y}$ be the interval computed right before the last intersection in the forward evaluation* $\mathtt{FE}(\mathbf{M}, [h])$ *(Definition 32). Then the following holds:*

$$\mathrm{w}(\tau_\mathbf{M} \cap \mathbf{y}) \geq r_\mathrm{f} * \mathrm{w}(\tau_\mathbf{M}) \quad \vee \quad \mathrm{w}(\tau_\mathbf{M} \cap \mathbf{y}) + d_\mathrm{f} \geq \mathrm{w}(\tau_\mathbf{M}).$$

- *Suppose that $\mathbf{N}$ is a parent of $\mathbf{M}$ and that the* **FBPD** *algorithm uses fixed interval forms of elementary operations in* $\mathtt{BP}(\mathbf{N}, \mathbf{M})$ *at all steps. Let $\mathbf{z}$ be the interval computed right before the last intersection in the backward propagation* $\mathtt{BP}(\mathbf{N}, \mathbf{M})$ *(Definition 38). Then the following holds:*

$$\mathrm{w}(\tau_\mathbf{M} \cap \mathbf{z}) \geq r_\mathrm{b} * \mathrm{w}(\tau_\mathbf{M}) \quad \vee \quad \mathrm{w}(\tau_\mathbf{M} \cap \mathbf{z}) + d_\mathrm{b} \geq \mathrm{w}(\tau_\mathbf{M}).$$

Proof. This follows directly from the discussion in Section 5.3. ■

6. Coordinating Constraint Propagation and Search

Next, we consider the issue of coordinating constraint propagation and search for solving NCSPs in the *branch-and-prune* framework – the most common framework for exhaustively solving NCSPs. The most widely used search algorithm is based on the bisection of domains, and is hence called *bisection search*. It is suitable for solving problems with isolated solutions. However, it is often inefficient for solving problems with a continuum of solutions. For such problems, therefore, we need more advanced search techniques. We consider the issue of integrating the **FBPD** algorithm into a generic branch-and-prune search algorithm, called **BnPSearch**, described in Algorithm 9.

Algorithm 9: The **BnPSearch** algorithm – a generic branch-and-prune search

Input: a CSP $\mathcal{P} \equiv (\mathcal{V}, \mathcal{D}, \mathcal{C})$.
Output: $\mathcal{L}_\forall$, $\mathcal{L}_\varepsilon$. ◀ Lists of inner and undiscernible boxes, respectively.
Construct the DAG representation, $D_\mathbf{G}$, of $\mathcal{P}$;
FPBD$(D_\mathbf{G}, \mathcal{C}, \mathcal{D})$; ◀ Prune the domains in $\mathcal{D}$ by using Algorithm 6.
if $\mathcal{D} = \emptyset$ **then return** `infeasible`;
if the domains in $\mathcal{D}$ are small enough **then**
    $\mathcal{L}_\varepsilon := \mathcal{L}_\varepsilon \cup \{(\mathcal{D}, \mathcal{C})\}$;
    return;
WaitingList $:= \{(\mathcal{D}, \mathcal{C})\}$;
while WaitingList $\neq \emptyset$ **do**
    Get a couple $(\mathcal{D}_0, \mathcal{C}_0)$ from WaitingList; ▼/* $\mathcal{D}_{i=\overline{1,k}} \subseteq \mathcal{D}_0$, $\mathcal{C}_{i=\overline{1,k}} \subseteq \mathcal{C}_0$. */
    Split the CSP $(\mathcal{V}, \mathcal{D}_0, \mathcal{C}_0)$ into sub-CSPs $\{(\mathcal{V}, \mathcal{D}_1, \mathcal{C}_1), \ldots, (\mathcal{V}, \mathcal{D}_k, \mathcal{C}_k)\}$;
    for $i := 1, \ldots, k$ **do**
        if $\mathcal{C}_i = \emptyset$ **then**
            $\mathcal{L}_\forall := \mathcal{L}_\forall \cup \{\mathcal{D}_i\}$; ◀ All points in $\mathcal{D}_i$ are solutions.
            continue for;
        FPBD$(D_\mathbf{G}, \mathcal{C}_i, \mathcal{D}_i)$; ◀ Prune the domains in $\mathcal{D}_i$ by using Algorithm 6.
        if $\mathcal{D}_i = \emptyset$ **then continue for**;
        if the domains in $\mathcal{D}_i$ are small enough **then**
            $\mathcal{L}_\varepsilon := \mathcal{L}_\varepsilon \cup \{(\mathcal{D}_i, \mathcal{C}_i)\}$; ◀ This CSP is not amenable to further splitting.
        WaitingList $:=$ WaitingList $\cup \{(\mathcal{D}_i, \mathcal{C}_i)\}$;

The **BnPSearch** algorithm produces two lists: $\mathcal{L}_\forall$, $\mathcal{L}_\varepsilon$. The first list, $\mathcal{L}_\forall$, consists of completely feasible domain boxes, called *inner boxes* or *feasible boxes*. That is, all points of a box in $\mathcal{L}_\forall$ are a solution of the problem. The second list, $\mathcal{L}_\varepsilon$, consists of subproblems, each consisting of a domain box and a set of running constraints. Each domain box of a subproblem in $\mathcal{L}_\varepsilon$ is canonical or smaller than the required precision ε. These domain boxes are called *undiscernible boxes*.

Owning to Theorem 40 and the finite nature of the floating-point number system, it is easy to prove that the branch-and-prune search in Algorithm 9 terminates after a finite number of iterations. Moreover, this search algorithm never discards any solution. Note that the **UCA6** and **UCA6$^+$** algorithms in [Silaghi et al. 2001; Vu et al. 2003] are specific instances of this generic search.

Table 1: A comparison of three constraint propagation techniques, **FBPD**, **BOX** and **HC4**, in solving NCSPs. In the section (a), the averages of the relative time ratios are 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. In the section (b), the averages of the relative ratios are taken over all the problems in the test cases T_4, T_5.

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	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 2: The averages of the relative time ratios are taken over the problems in each test case.

Propagator ▼	(a) Isolated Solutions			(b) Continuum of Solutions	
	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	13.45	11.55	31.85
HC4	94.42	691.24	68.17	191.86	651.31

7. Experiments

We have carried out experiments on the **FBPD** algorithm and two other well-known state-of-the-art interval constraint propagation techniques. The first propagation technique is a variant of box consistency [Benhamou et al. 1994] implemented in a commercial product, ILOG Solver (v6.0), hereafter denoted as **BOX**. The second constraint propagation technique is the **HC4** algorithm (see Section 2.3.2). The experiments are carried out on 33 problems, which are impartially chosen and divided into five test cases, to analyze the empirical results:

- The test case T_1 (see Section A.1) consists of eight easy problems with isolated solutions. These problems are solvable in short time by the search using all three propagators.

- The test case T_2 (see Section A.2) consists of four problems of moderate difficulty with isolated solutions. These problems are solvable by the search using **FBPD** and **BOX** and cause the search using **HC4** being out of time without reaching 10^6 splits.

Table 3: The overrun ratios for the test case T_1. (An overrun ratio greater than 1 would satisfy the requirements of applications.)

Problem ►	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**

- The test case T_3 (see Section A.3) consists of eight hard problems with isolated solutions. These problems cause the search using **FBPD** to stop due to running more than 10^6 splits, cause the search using **HC4** to be out of time without reaching 10^6 splits, and cause the search using **BOX** either to be out of time or to stop due to running more than 10^6 splits.[4]

- The test case T_4 (see Section A.4) consists of seven easy problems with a continuum of solutions. These problems are solvable in short time at the predefined precision 10^{-2}.

- The test case T_5 (see Section A.5) consists of six hard problems with a continuum of solutions. These problems are solvable in short time at the predefined precision 10^{-1}.

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 that are out of time in the next result analysis*** favor of slow techniques). For the first three test cases, the precision is 10^{-4}, and the search is done by bisection. For the last two test cases, the search is performed using the **UCA6** [Silaghi et al. 2001], algorithm for inequalities. The comparison of the interval propagation techniques is based on the following measures:

- *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 splits/iterations:* The number of splits in search needed to solve the problems. The relative ratio of the number of splits used by each propagator to that of **FBPD** is called the *relative iteration ratio.*

4. **FBPD** essentially works at the node level. Evaluation/propagation procedures can therefore be run on selected nodes rather than on the entire graph. This enables the use of different interval forms at different steps of a propagation procedure. Such an approach was tested in [Vu et al. 2004a]. This extension of FBPD was then able to solve 6 problems out of 8 in the test case T_3

- *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 lower the relative ratio is, the better the performance/quality is; and the higher the inner volume ratio is, the better the quality is.

The overviews of results in our experiments are given in Table 1 and Table 2. In Table 3, 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 precision, 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 1 to 2 orders of magnitude or more in speed. For problems with a continuum of solutions, **FBPD** has roughly the same quality with respect to enclosure properties. For isolated solutions, very narrow boxes are produced by any technique in comparison to the required precision. However, the new technique is about 1.1–2.0 times less tight than the other techniques in the measure of reduction per dimension (which hardly matters in applications).

The gain in performance is more important for under-constrained problems than for well-constrained ones.

8. Conclusion

We propose a new constraint propagation technique, called **FBPD**, which makes the fundamental framework of interval analysis on DAGs [Schichl and Neumaier 2005] efficient and practical for numerical constraint propagation. We also propose a method to coordinate constraint propagation (**FBPD**) and exhaustive search on partial DAG representations, where only one DAG for each problem is needed for the whole solution process. The experiments, carried out on various problems, show that the new approach can outperform previously available propagation techniques by 1 to 2 orders of magnitude or more in speed, while being roughly of same quality with respect to enclosure properties. Moreover, **FBPD** essentially works at the node level. Evaluation and propagation procedures can therefore be run on selected nodes rather than on the entire graph. This enables the use of different enclosure techniques at different steps of the propagation process opens up promising perspectives [Vu et al. 2004a].

Acknowledgments

Support for this research was partially provided by the European Commission and the Swiss Federal Education and Science Office (OFES) through the COCONUT project (MIST-2000-26063). We would like to thank ILOG for the software licenses of ILOG Solver used in the COCONUT project, and thank the COCONUT team of the University of Nantes for the **HC4** source code. We also thank Professor Arnold Neumaier at the University of Vienna (Austria) for fruitful discussions and very valuable input.

References

Alefeld, G. and Herzberger, J. (1983). *Introduction to Interval Computations*. Academic Press, New York, NY.

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

Benhamou, F., McAllester, D., and Van Hentenryck, P. (1994). CLP(Intervals) Revisited. In *Proceedings of the International Logic Programming Symposium*, pages 109–123.

Benhamou, F. and Older, W. J. (1992). Applying Interval Arithmetic to Real, Integer and Boolean Constraints. Technical Report BNR Technical Report, Bell Northern Research, Ontario, Canada.

Benhamou, F. and Older, W. J. (1997). Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, pages 32–81. Extension of a technical report of Bell Northern Research, Canada, 1992.

Goldberg, D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1):5–48.

Hansen, E. R. and Walster, G. W. (2004). *Global Optimization Using Interval Analysis*. Marcel Dekker, second edition.

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

Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001). *Applied Interval Analysis*. Springer, first edition.

Lhomme, O. (1993). Consistency Techniques for Numeric CSPs. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 232–238.

Lottaz, C. (2000). *Collaborative Design using Solution Spaces*. PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland.

Mackworth, A. K. (1977). Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118.

McCormick, G. P. (1976). Computability of Global Solutions to Factorable Nonconvex Programs: Part I – Convex Underestimating Problems. *Mathematical Programming*, 10:147–175.

McCormick, G. P. (1983). *Nonlinear Programming: Theory, Algorithms and Applications*. John Wiley & Sons.

Montanari, U. (1974). Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7:95–132.

Moore, R. E. (1966). *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ.

Moore, R. E. (1979). *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics. Philadelphia.

Neumaier, A. (1990). *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge.

Sam-Haroud, D. (1995). *Constraint Consistency Techniques for Continuous Domains*. PhD thesis, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland.

Schichl, H. (2003). Mathematical Modeling and Global Optimization. Habilitation thesis, Faculty of Mathematics, University of Vienna, Autralia.

Schichl, H. and Neumaier, A. (2005). Interval Analysis on Directed Acyclic Graphs for Global Optimization. *Journal of Global Optimization*, 33:541–562.

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

Singh, S., Watson, B., and Srivastava, P. (1997). *Fixed Point Theory and Best Approximation: The KKM-map Principle*. Kluwer Academic Publishers, Dordrecht.

Van Hentenryck, P. (1997). Numerica: A Modeling Language for Global Optimization. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*.

Vu, X.-H., Sam-Haroud, D., and Faltings, B. (2004a). Combining Multiple Inclusion Representations in Numerical Constraint Propagation. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pages 458–467, Florida, USA. IEEE Computer Society Press.

Vu, X.-H., Sam-Haroud, D., and Silaghi, M.-C. (2003). Numerical Constraint Satisfaction Problems with Non-isolated Solutions. In *Global Optimization and Constraint Satisfaction: First International Workshop on Global Constraint Optimization and Constraint Satisfaction, COCOS 2002*, volume LNCS 2861, pages 194–210, Valbonne-Sophia Antipolis, France. Springer-Verlag.

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

Walster, G. W., Hansen, E. R., and Pryce, J. D. (2000). Extended Real Intervals and the Topological Closure of Extended Real Relations. Technical report, Sun Microsystems. `http://wwws.sun.com/software/sundev/whitepapers/extended-real.pdf`.

Waltz, D. L. (1972). Generating Semantic Descriptions from Drawings of Scenes with Shadows. Technical report, Massachusetts Institute of Technology, USA.

Waltz, D. L. (1975). *The Psychology of Computer Vision*, chapter Understanding Line Drawings of Scenes with Shadows, pages 19–91. McGraw Hill, New York.

Appendix A. Numerical Benchmarks

A.1 Test Case T_1: Problems with Isolated Solutions

A.1.1 Problem BIF3

A bifurcation problem:

$$\begin{cases} 5x^9 - 6x^5y^2 + xy^4 + 2xz = 0; \\ -2x^6y + 2x^2y^3 + 2yz = 0; \\ x^2 + y^2 = 0.265625; \end{cases}$$

where x, y, z in $[-10^8, 10^8]$.

A.1.2 Problem ECO5

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4)x_5 - 1 & = & 0; \\ (x_2 + x_1x_3 + x_2x_4)x_5 - 2 & = & 0; \\ (x_3 + x_1x_4)x_5 - 3 & = & 0; \\ x_4x_5 - 4 & = & 0; \\ x_1 + x_2 + x_3 + x_4 + 1 & = & 0; \end{cases}$$

where $x_1, \ldots, x_5$ in $[-10, 10]$.

A.1.3 Problem ECO6

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5)x_6 - 1 & = & 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5)x_6 - 2 & = & 0; \\ (x_3 + x_1x_4 + x_2x_5)x_6 - 3 & = & 0; \\ (x_4 + x_1x_5)x_6 - 4 & = & 0; \\ x_5x_6 - 5 & = & 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + 1 & = & 0; \end{cases}$$

where $x_1, \ldots, x_6$ in $[-10, 10]$.

A.1.4 Problem ECO7

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6)x_7 - 1 & = & 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6)x_7 - 2 & = & 0; \\ (x_3 + x_1x_4 + x_2x_5 + x_3x_6)x_7 - 3 & = & 0; \\ (x_4 + x_1x_5 + x_2x_6)x_7 - 4 & = & 0; \\ (x_5 + x_1x_6)x_7 - 5 & = & 0; \\ x_6x_7 - 6 & = & 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 1 & = & 0; \end{cases}$$

where $x_1, \ldots, x_7$ in $[-10, 10]$.

A.1.5 Problem **ECO8**

An economic problem:

$$\begin{cases} (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7)x_8 - 1 & = & 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6 + x_5x_7)x_8 - 2 & = & 0; \\ (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6)x_7 - 2 & = & 0; \\ (x_4 + x_1x_5 + x_2x_6 + x_3x_7)x_8 - 4 & = & 0; \\ (x_5 + x_1x_6 + x_2x_7)x_8 - 5 & = & 0; \\ (x_6 + x_1x_7)x_8 - 6 & = & 0; \\ x_7x_8 - 7 & = & 0; \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + 1 & = & 0; \end{cases}$$

where $x_1, \ldots, x_8$ in $[-10, 10]$.

A.1.6 Problem **NEU6**

A neurophysiology problem:

$$\begin{cases} x_1^2 + x_3^2 & = & 1; \\ x_2^2 + x_4^2 & = & 1; \\ x_5x_1^3 + x_6x_2^3 & = & 5; \\ x_5x_1x_3^2 + x_6x_4^2x_2 & = & 4; \\ x_5x_3^3 + x_6x_4^3 & = & 3; \\ x_5x_1^2x_3 + x_6x_2^2x_4 & = & 2; \\ x_1 & \geq & x_2; \\ x_1 & \geq & 0; \\ x_2 & \geq & 0; \end{cases}$$

where $x_1, \ldots, x_6$ in $[-100, 100]$.

A.1.7 Problem **REI3**

A neurophysiology problem:

$$\begin{cases} x^2 - y^2 + z^2 & = & 0.5; \\ x^3 - y^3 + z^3 & = & 0.5; \\ x^4 - y^4 + z^4 & = & 0.5; \\ 2xy + 6y^2 + 2yz - 2x - 4y - 2z + 1 & = & 0; \end{cases}$$

where x, y, z in $[-10, 10]$.

A.1.8 Problem **WIN3**

A neurophysiology problem:

$$\begin{cases} 4xz - 4xy^2 - 16x^2 - 1 & = & 0; \\ 2y^2z + 4x + 1 & = & 0; \\ 2x^2z + 2y^2 + x & = & 0; \\ 2xy + 6y^2 + 2yz - 2x - 4y - 2z + 1 & = & 0; \end{cases}$$

where x, y, z in $[-10^5, 10^5]$.

A.2 Test Case T_2: Problems with Isolated Solutions

A.2.1 Problem **CYC5**

A cyclic problem:

$$\begin{cases} a+b+c+d+e & = & 0; \\ ab+bc+cd+de+ea & = & 0; \\ abc+bcd+cde+dea+eab & = & 0; \\ abcd+bcde+cdea+deab+eabc & = & 0; \\ abcde-1 & = & 0; \end{cases}$$

where a, b, c, d, e in $[-10, 10]$.

A.2.2 Problem **GS5.1**

A Gough Steward problem:

$$\begin{cases} x_1^2+y_1^2+z_1^2 & = & 31, \\ x_2^2+y_2^2+z_2^2 & = & 39, \\ x_3^2+y_3^2+z_3^2 & = & 29, \\ x_1x_2+y_1y_2+z_1z_2+6x_1-6x_2 & = & 51, \\ x_1x_3+y_1y_3+z_1z_3+7x_1-2y_1-7x_3+2y_3 & = & 50, \\ x_2x_3+y_2y_3+z_2z_3+x_2-2y_2-x_3+2y_3 & = & 34, \\ -12x_1+15y_1-10x_2-25y_2+18x_3+18y_3 & = & -32, \\ -14x_1+35y_1-36x_2-45y_2+30x_3+18y_3 & = & 8, \\ 2x_1+2y_1-14x_2-2y_2+8x_3-y_3 & = & 20, \end{cases}$$

where $x_1 \in [0.00; 5.57]$, $y_1 \in [0.00, 2.70]$, $z_1 \in [0.00, 5.57]$, $x_2 \in [-6.25, 0.00]$, $y_2 \in [-2.00, 0.00]$, $z_2 \in [0.00, 6.25]$, $x_3 \in [-5.39, -1.00]$, $y_3 \in [-5.39, 0.00]$, $z_3 \in [0.00, 5.39]$.

A.2.3 Problem **KOL2**

Kolev's benchmark:

$$\begin{cases} ((4x_3+3x_6)x_3+2x_5)x_3+x_4 & = & 0, \\ ((4x_2+3x_6)x_2+2x_5)x_2+x_4 & = & 0, \\ ((4x_1+3x_6)x_1+2x_5)x_1+x_4 & = & 0, \\ x_4+x_5+x_6+1 & = & 0, \\ (((x_2+x_6)x_2+x_5)x_2+x_4)x_2+(((x_3+x_6)x_3+x_5)x_3+x_4)x_3 & = & 0, \\ (((x_1+x_6)x_1+x_5)x_1+x_4)x_1+(((x_2+x_6)x_2+x_5)x_2+x_4)x_3 & = & 0, \end{cases}$$

where $x_1 \in [0.0333, 0.2173]$, $x_2 \in [0.4000, 0.6000]$, $x_3 \in [0.7826, 0.9666]$, $x_4 \in [-0.3071, -0.1071]$, $x_5 \in [1.1071, 1.3071]$, $x_6 \in [-2.1000, -1.9000]$.

A.2.4 Problem **YAM60**

The Yama160 problem:

$$(n+1)^2 x_{i-1} - 2(n+1)^2 x_i + (n+1)^2 x_{i+1} + e^{x_i} = 0, \quad (\text{for } i = 1, \ldots, n),$$

where $n = 60$, $x_0 = x_{n+1} = 0$, and $x_i \in [-10, 10]$ (for $i = 1, \ldots, n$),

A.3 Test Case T_3: Problems with Isolated Solutions

A.3.1 Problem **CAP4**

A Caprasse problem:

$$\begin{cases} y^2z + 2xyt - 2x - z & = & 0; \\ -x^3z + 4xy^2z + 4x^2yt + 2y^3t + 4x^2 - 10y^2 + 4xz - 10yt + 2 & = & 0; \\ 2yzt + xt^2 - x - 2z & = & 0; \\ -xz^3 + 4yz^2t + 4xzt^2 + 2yt^3 + 4xz + 4z^2 - 10yt - 10t^2 + 2 & = & 0; \end{cases}$$

where x, y, z, t in $\mathbb{R}$.

A.3.2 Problem **DID9**

A Didrit problem:

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 = 31; \\ x_2^2 + y_2^2 + z_2^2 = 39; \\ x_3^2 + y_3^2 + z_3^2 = 29; \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 = 51; \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 = 50; \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 = 34; \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 = -32; \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 = 8; \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 = 20; \end{cases}$$

where x_i, y_i, z_i in $[-10, 10]$ for $i = 1, 2, 3$.

A.3.3 Problem **GS5.0**

A Gough Steward problem:

$$\begin{cases} x_1^2 + y_1^2 + z_1^2 & = & 31, \\ x_2^2 + y_2^2 + z_2^2 & = & 39, \\ x_3^2 + y_3^2 + z_3^2 & = & 29, \\ x_1x_2 + y_1y_2 + z_1z_2 + 6x_1 - 6x_2 & = & 51, \\ x_1x_3 + y_1y_3 + z_1z_3 + 7x_1 - 2y_1 - 7x_3 + 2y_3 & = & 50, \\ x_2x_3 + y_2y_3 + z_2z_3 + x_2 - 2y_2 - x_3 + 2y_3 & = & 34, \\ -12x_1 + 15y_1 - 10x_2 - 25y_2 + 18x_3 + 18y_3 & = & -32, \\ -14x_1 + 35y_1 - 36x_2 - 45y_2 + 30x_3 + 18y_3 & = & 8, \\ 2x_1 + 2y_1 - 14x_2 - 2y_2 + 8x_3 - y_3 & = & 20, \end{cases}$$

where $x_1 \in [-2.00; 5.57]$, $y_1 \in [-5.57, 2.70]$, $z_1 \in [0.00, 5.57]$, $x_2 \in [-6.25, 1.30]$, $y_2 \in [-6.25, 2.70]$, $z_2 \in [-2.00, 6.25]$, $x_3 \in [-5.39, 0.70]$, $y_3 \in [-5.39, 3.11]$, $z_3 \in [-3.61, 5.39]$.

A.3.4 Problem **KAT8**

A Katsura problem:

$$\begin{cases} -x_1 + 2x_8^2 + 2x_7^2 + 2x_6^2 + 2x_5^2 + 2x_4^2 + 2x_3^2 + 2x_2^2 + x_1^2 & = & 0; \\ -x_2 + 2x_8x_7 + 2x_7x_6 + 2x_6x_5 + 2x_5x_4 + 2x_4x_3 + 2x_3x_2 + 2x_2x_1 & = & 0; \\ -x_3 + 2x_8x_6 + 2x_7x_5 + 2x_6x_4 + 2x_5x_3 + 2x_4x_2 + 2x_3x_1 + x_2^2 & = & 0; \\ -x_4 + 2x_8x_5 + 2x_7x_4 + 2x_6x_3 + 2x_5x_2 + 2x_4x_1 + 2x_3x_2 & = & 0; \\ -x_5 + 2x_8x_4 + 2x_7x_3 + 2x_6x_2 + 2x_5x_1 + 2x_4x_2 + x_3^2 & = & 0; \\ -x_6 + 2x_8x_3 + 2x_7x_2 + 2x_6x_1 + 2x_5x_2 + 2x_4x_3 & = & 0; \\ -x_7 + 2x_8x_2 + 2x_7x_1 + 2x_6x_2 + 2x_5x_3 + x_4^2 & = & 0; \\ -1 + 2x_8 + 2x_7 + 2x_6 + 2x_5 + 2x_4 + 2x_3 + 2x_2 + x_1 & = & 0; \end{cases}$$

where $x_1, \ldots, x_8$ in $[-10, 10]$.

A.3.5 Problem **KIN9**

A kinematics problem:

$$\begin{cases} z_1^2 + z_2^2 + z_3^2 - 12z_1 - 68 & = & 0; \\ z_4^2 + z_5^2 + z_6^2 - 12z_5 - 68 & = & 0; \\ z_7^2 + z_8^2 + z_9^2 - 24z_8 - 12z_9 + 100 & = & 0; \\ z_1z_4 + z_2z_5 + z_3z_6 - 6z_1 - 6z_5 - 52 & = & 0; \\ z_1z_7 + z_2z_8 + z_3z_9 - 6z_1 - 12z_8 - 6z_9 + 64 & = & 0; \\ z_4z_7 + z_5z_8 + z_6z_9 - 6z_5 - 12z_8 - 6z_9 + 32 & = & 0; \\ 2z_2 + 2z_3 - z_4 - z_5 - 2z_6 - z_7 - z_9 + 18 & = & 0; \\ z_1 + z_2 + 2z_3 + 2z_4 + 2z_6 - 2z_7 + z_8 - z_9 - 38 & = & 0; \\ z_1 + z_3 - 2z_4 + z_5 - z_6 + 2z_7 - 2z_8 + 8 & = & 0; \end{cases}$$

where $z_1, \ldots, z_9$ in $[-1000, 1000]$.

A.3.6 Problem **REI4**

A Reinmer system:

$$\begin{cases} x^2 - y^2 + z^2 - t^2 = 0.5; \\ x^3 - y^3 + z^3 - t^3 = 0.5; \\ x^4 - y^4 + z^4 - t^4 = 0.5; \\ x^5 - y^5 + z^5 - t^5 = 0.5; \end{cases}$$

where x, y, z, t in $[-10, 10]$.

A.3.7 Problem **REI5**

A Reinmer system:

$$\begin{cases} -1 + 2x_1^2 - 2x_2^2 + 2x_3^2 - 2x_4^2 + 2x_5^2 = 0; \\ -1 + 2x_1^3 - 2x_2^3 + 2x_3^3 - 2x_4^3 + 2x_5^3 = 0; \\ -1 + 2x_1^4 - 2x_2^4 + 2x_3^4 - 2x_4^4 + 2x_5^4 = 0; \\ -1 + 2x_1^5 - 2x_2^5 + 2x_3^5 - 2x_4^5 + 2x_5^5 = 0; \\ -1 + 2x_1^6 - 2x_2^6 + 2x_3^6 - 2x_4^6 + 2x_5^6 = 0; \end{cases}$$

where $x_1, \ldots, x_5$ in $[-1, 1]$.

A.3.8 Problem **REI6**

A Reinmer system:

$$\langle -0.5 + \sum_{i=1}^{n} (-1)^{i+1} x_i^k = 0 \ (k = 1, \ldots, n); \ n = 6, \ x_i \in [-1, 1] \ (\text{for } i = 1, \ldots, n) \rangle$$

A.4 Test Case T_4: Problems with Continuums of Solutions

A.4.1 Problem **F2.2**

Tricuspoid and Circle:

$$\begin{cases} (x^2 + y^2 + 12x + 9)^2 \leq 4(2x + 3)^3; \\ x^2 + y^2 \geq 2; \end{cases}$$

where x, y in $[-2, 2]$.

A.4.2 Problem **F2.3**

Foliumd, Circle, and Trifolium:

$$\begin{cases} x^3 + y^3 \geq 3xy; \\ x^2 + y^2 \geq 0.1; \\ (x^2 + y^2)(y^2 + x(x + 1)) \leq 4xy^2; \end{cases}$$

where x, y in $[-3, 3]$.

A.4.3 Problem **S04**

Circle:

$$\langle x^2 + y^2 \leq 1; \ x, y \in [-2, 2] \rangle$$

A.4.4 Problem **S05**

$$\langle \frac{x}{\sqrt{(y - 5)^2 + 1}} \leq 1; \ x, y \in [1, 10] \rangle$$

A.4.5 Problem **S06**

$$\langle \frac{12y}{\sqrt{(x - 12)^2 + y^2}} \leq 10; \ x \in [-50, 50], \ y \in [0, 50] \rangle$$

A.4.6 Problem **S07**

$$\langle x^2 + y^2 \geq 20; \ x^2 + y^2 \leq 50; \ x \in [-50, 50], \ y \in [0, 50] \rangle$$

A.4.7 Problem **WP**

A Kinematic Pair (of a wheel and a pawl):

$$\langle 20 \leq \sqrt{x^2 + y^2} \leq 50, \ \frac{12y}{\sqrt{(x - 12)^2 + y^2}} \leq 10; \ x \in [-50, 50], \ y \in [0, 50] \rangle$$

A.5 Test Case T_5: Problems with Continuums of Solutions

A.5.1 Problem **G1.1**

$$\begin{cases} x_1^2 + 0.5x_2 + 2(x_3 - 6) \geq 0; \\ x_1^2 + x_2^2 + x_3^2 \leq 25; \end{cases}$$

where x_1, x_2, x_3 in $[-8, 8]$.

A.5.2 Problem **G1.1**

$$\begin{cases} x_1^2 + 0.5x_2 + 2(x_3 - 3) \geq 0; \\ x_1^2 + x_2^2 + x_3^2 \leq 25; \end{cases}$$

where x_1, x_2, x_3 in $[-8, 8]$.

A.5.3 Problem **H1.1**

$$\begin{cases} x_1^2 + x_2^2 + x_3^2 \leq 9; \\ (x_1 - 0.5)^2 + (x_2 - 1)^2 + x_3^2 \geq 4; \\ x_1^2 + (x_2 - 0.2)^2 \geq x_3; \end{cases}$$

where x_1, x_2, x_3 in $[-4, 4]$.

A.5.4 Problem **P1.4**

$$\begin{cases} x^2 + y^2 + z^2 <= 4; \\ (x - 2)^2 + y^2 + z^2 >= 4; \end{cases}$$

where x, y, z in $[-4, 4]$.

A.5.5 Problem **P2**

$$\begin{cases} x^2 \leq y, \\ \ln y + 1 \geq z, \\ xz \leq 1, \end{cases}$$

where $x \in [0, 15],\ y \in [1, 200],\ z \in [-10, 10]$.

A.5.6 Problem **P3**

$$\begin{cases} x^2 \leq y, \\ \ln y + 1 \geq z, \\ xz \leq 1, \\ x^{3/2} + \ln(1.5z + 1) \leq y + 1, \end{cases}$$

where $x \in [0, 15],\ y \in [1, 200],\ z \in [0, 10]$.