Algorithm 852: RealPaver: An Interval Solver Using Constraint Satisfaction Techniques

LAURENT GRANVILLIERS and FRÉDÉRIC BENHAMOU
University of Nantes

RealPaver is an interval software for modeling and solving nonlinear systems. Reliable approximations of continuous or discrete solution sets are computed using Cartesian products of intervals. Systems are given by sets of equations or inequality constraints over integer and real variables. Moreover, they may have different natures, being square or nonsquare, sparse or dense, linear, polynomial, or involving transcendental functions.

The modeling language permits stating constraint models and tuning parameters of solving algorithms which efficiently combine interval methods and constraint satisfaction techniques. Several consistency techniques (box, hull, and 3B) are implemented. The distribution includes C sources, executables for different machine architectures, documentation, and benchmarks. The portability is ensured by the GNU C compiler.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; G.1.0 [**Numerical Analysis**]: General—*Interval arithmetic*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Reliability and robustness*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interval arithmetic, constraint satisfaction, nonlinear system, local consistency, interval Newton.

1. INTRODUCTION

The software package RealPaver [Granvilliers 2003] implements a modeling language and some interval-based algorithms to process systems of nonlinear constraints over the real numbers. Given a vector $(x_1, \dots, x_n) \in \mathbb{R}^n$ of unknowns, a constraint system $(\mathcal{C}, \mathbf{x})$ is defined by a set $\mathcal{C} = \{c_1, \dots, c_p\}$ of constraints and a bounded domain $\mathbf{x} = \boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n$ as follows:

$$\begin{cases} c_i : f_i(x_1, \dots, x_n) = 0, & i = 1, \dots, m, \\ c_j : f_j(x_1, \dots, x_n) \leq 0, & j = m+1, \dots, p, \\ x_k \in \boldsymbol{x}_k = \{r \in \mathbb{R} \mid a_k \leq r \leq b_k\}, & k = 1, \dots, n. \end{cases} \quad (1)$$

Authors' address: L. Granvilliers, F. Benhamou, LINA, University of Nantes, 2 rue de la Houssinière, BP 92208, F-44322 Nantes Cedex 3, France; email: {Laurent.Granvilliers,Frederic.Benhamou}@univ-nantes.fr
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2006 ACM 0098-3500/06/0300-0138 $5.00

Each constraint is an equation or an inequality constraint involving nonlinear analytic expressions which do not need to be differentiable. Each variable lies in a closed real interval. The Cartesian product of variable domains $\mathbf{x}$ is called a box. The solution set is the set of tuples in $\mathbf{x}$ that satisfy all the constraints from $\mathcal{C}$.

The purpose of interval-based algorithms is to generate a set of n-dimensional boxes whose union encloses the solution set. Boxes are generated from the initial domain using a branch-and-prune algorithm, which is an iterative method that alternates pruning and branching steps until reaching some fixed precision. The pruning step aims at reducing a box by eliminating inconsistent values. The branching step splits a box into a set of smaller boxes.

The pruning step in RealPaver merges different techniques. Constraint satisfaction techniques [Van Hentenryck et al. 1997; Granvilliers et al. 2001] are used to narrow down the domains by removing locally inconsistent intervals containing no solution of some constraint. The interval Newton method [Neumaier 1990] is more specifically applied to square systems of equations, if such a system can be extracted from a model. The interval Newton method can be decomposed in two steps: The first step generates a linear relaxation of the system through Taylor expansions; the second step implements the interval Gauss-Seidel algorithm to handle the linear interval system. The cooperative pruning algorithm has been shown in Granvilliers [2001] to compete with state-of-the-art solvers based on interval or continuation methods.

The modeling language looks like the one of AMPL [Fourer et al. 2003]. For instance, consider a Gough-Stewart platform modeling the kinematics of parallel robots [Stewart 1965]. The complete model follows, just stating the set of unknowns and their domains, and the set of constraints. The variables define the position and orientation of the platform according to the base.

```
/* RealPaver model: Gough-Stewart platform */
Variables   x1 in [-2,5.57]   , x2 in [-6.25,1.3], x3 in [-5.39,0.7] ,
            y1 in [-5.57,2.7], y2 in [-6.25,2.7], y3 in [-5.39,3.11],
            z1 in [0,5.57]    , z2 in [-2,6.25]   , z3 in [-3.61,5.39];

Constraints x1^2 + y1^2 + z1^2 = 31,
            x2^2 + y2^2 + z2^2 = 39,
            x3^2 + y3^2 + z3^2 = 29,
            x1*x2 + y1*y2 + z1*z2 + 6*x1 - 6*x2 = 51,
            x1*x3 + y1*y3 + z1*z3 + 7*x1 - 2*y1 - 7*x3 + 2*y3 = 50,
            x2*x3 + y2*y3 + z1*z3 + x2 - 2*y2 - x3 + 2*y3 = 34,
            -12*x1 + 15*y1 - 10*x2 - 25*y2 + 18*x3 + 18*y3 = -32,
            -14*x1 + 35*y1 - 36*x2 - 45*y2 + 30*x3 + 18*y3 = 8,
            2*x1 + 2*y1 - 14*x2 - 2*y2 + 8*x3 - y3 = 20;
```

For this problem, RealPaver computes a set of boxes, each of which encloses an isolated solution. The first computed box is presented below, where an interval is printed as its midpoint and reliable error bounds:

```
x1 =  1.006631120412478  + [-5.939e-10, +5.939e-10]
x2 = -2.993384623694497  + [-1.34e-09,  +1.34e-09]
x3 = -2.974335204662014  + [-1.42e-09,  +1.42e-09]
y1 = -0.1790667191214789 + [-2.038e-09, +2.038e-09]
y2 = -0.1729309431572662 + [-4.073e-10, +4.073e-10]
y3 =  0.1136937833229701 + [-6.671e-11, +6.671e-11]
z1 =  5.473082211836417  + [-3.268e-12, +3.268e-12]
z2 =  5.478114948002476  + [-8.574e-12, +8.574e-12]
z3 =  4.487806124816657  + [-1.5e-11,   +1.5e-11]
```

`RealPaver` is used in several application domains. In automatic control, the main goal is to prove identifiability, robustness, or stability [Jaulin et al. 2001]. Constraint-aided conceptual design represents another application area [O'Sullivan 1999]. The main improvements with respect to classical simulation codes are twofold: The ability of constraint solving algorithms to deal with partial information; and the early integration of physical laws, design rules, quality criteria, and design objectives into a single model. As a consequence, the computed concepts necessarily verify constraints, which may reduce the conceptual design phase. Data-fitting applications can also be handled [Schittkowski 2002]. Rather than computing a best-fitting model, compact representations of the set of consistent values of model parameters are determined which may help in decision making.

`RealPaver` is written in C (13000 lines). The parser of the modeling language is generated by the GNU tools `flex` and `bison` [Free Software Foundation 2002]. The software is known to compile on several different machine architectures. The portability is ensured by the GNU C compiler. Moreover, specific codes for rounding operations in floating-point arithmetic have been implemented.

1.1 Related Interval Software

Several interval softwares have been developed, most of them being referenced on the interval computations webpage [Berleant and Kreinovich 2003]. We mention some of these products here.

Interval-based constraint satisfaction techniques have been implemented in several constraint logic programming (CLP) systems, like `Eclipse` [Meier and Schimpf 1993] and `Prolog IV` [PrologIA 1996]. Constraints are embedded into CLP programs using Horn clauses. The operational semantics is based on term unification and constraint solving techniques.

The integration of constraint satisfaction techniques and interval methods has been pioneered by Pascal Van Hentenryck and his colleagues, the authors of the modeling language for global optimization `Numerica` [Van Hentenryck et al. 1997]. `Numerica` is not currently available, However, some of its constraint solving functionalities have been implemented in the commercial object-oriented library `ILOG Solver` [ILOG 2001]. A comparison of `RealPaver` and `Numerica` can be found in Granvilliers [2001]. `Unicalc` [Semenov et al. 1997] is a commercial software dedicated to the solving of systems containing equations, inequality constraints, and implications with integer and real variables. The resolution procedure is based on constraint satisfaction techniques. `Alias` [Merlet 2002] is a C++/Maple library for solving nonlinear systems of equations and optimization problems. Several constraint satisfaction, interval, and algebraic algorithms have been implemented.

What we believe as the main advantages of `RealPaver` over these systems are a fine-grained pruning algorithm based on advanced constraint satisfaction techniques. Moreover, `RealPaver` is able to tackle infinite solution sets with reliability. Some of the capabilities of the software will be illustrated in Section 3.

Several systems implement interval methods, like `GlobSol` [Kearfott 1996], or `COSY` [Berz and Makino 2002]. `COSY` is based on Taylor models to compute precise approximations of real functions. The solving engine of `GlobSol` implements a cooperative strategy which uses elementary constraint satisfaction techniques for constrained global optimization.

1.2 Outline

Section 2 presents interval arithmetic and consistency techniques, and describes the generic interval branch-and-prune algorithm implemented in `RealPaver`. In Section 3, we illustrate the use of the modeling language and show different executions of the software. Section 4 discusses solving strategies and their definitions through the modeling language. The package distribution is presented in Section 5, and perspectives about future releases are given in Section 6.

2. INTERVAL BRANCH-AND-PRUNE ALGORITHMS

The solving engine of `RealPaver` is designed to approximate the solution set of a nonlinear system by a union of boxes. Computations are validated in the sense that the approximation contains all the solutions.

A generic branch-and-prune algorithm is presented in Table I. Given an initial box $\mathbf{x}$, three sets of boxes are derived. $\mathcal{I}$ is the set of inner boxes that are proved to be included in the solution set. Boxes that are proved to be solution-free are eliminated. Sets $\mathcal{L}$ and $\mathcal{O}$ contain outer boxes, which are boxes that may contain solutions. The principle of branch-and-prune algorithms is to narrow down boxes from $\mathcal{L}$ until it becomes empty. The output is made of the inner approximation $\mathcal{I}$ and the union of outer boxes $\mathcal{O}$ whose widths are under a given precision ε.

Boxes are processed by interval computations. Box pruning is a reduction procedure removing values that do not belong to the solution set. In particular, a box is eliminated if at least one constraint is proved to be violated for all points from the box. Box splitting is a search procedure that generates unions of smaller boxes. In the following, specific interval reasonings that can be implemented in the branch-and-prune algorithm are described. For a thorough presentation, the reader is referred to Alefeld and Herzberger [1983] for interval arithmetic, to Neumaier [1990] for interval analysis, to Van Hentenryck et al. [1997] for constraint satisfaction techniques, and to Granvilliers [2001] for solving strategies.

Table I. Generic Branch-and-Prune Algorithm

```
1 BranchAndPrune($\mathcal{C}$: set of constraints, $\mathbf{x}$: box, $\varepsilon$ : positive floating-point number)
2   returns Union of boxes
3 begin
4   $\mathcal{I} := \varnothing$                                   % union of inner boxes
5   $\mathcal{O} := \varnothing$                                   % union of outer boxes
6   $\mathcal{L} := \{\mathbf{x}\}$                                 % set of boxes to be processed
7   while $\mathcal{L} \neq \varnothing$ do
8     $\mathbf{y} :=$ prune(extract($\mathcal{L}$), $\mathcal{C}$)                % pruning step
9     if $\mathbf{y} \neq \varnothing$ then
10      if $\mathbf{y}$ is included in the solution set then   % inner test
11        $\mathcal{I} := \mathcal{I} \cup \{\mathbf{y}\}$
12      elsif $\mathbf{y}$ is more precise than $\varepsilon$ then        % precision test
13        $\mathcal{O} := \mathcal{O} \cup \{\mathbf{y}\}$
14      else
15        $\mathcal{L} := \mathcal{L} \cup$ split($\mathbf{y}, \varepsilon$)              % splitting step
16      fi
17    fi
18  od
19  return $\mathcal{I} \cup \mathcal{O}$                            % reliable approximation
20 end
```

2.1 Interval Numbers

We consider the following sets: the set $\mathbb{R}$ of real numbers including the infinities, the finite set $\mathbb{F}$ of IEEE floating-point numbers [IEEE 1985] and the finite set $\mathbb{I}$ of closed intervals bounded by floating-point numbers. Every interval $\boldsymbol{x} \in \mathbb{I}$ is denoted by $[\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$ and is defined as the set of real numbers $\{r \in \mathbb{R} \mid \underline{\boldsymbol{x}} \leq r \leq \overline{\boldsymbol{x}}\}$. Interval computations heavily use the following rounding operations over the reals:

$$\begin{array}{l} \lfloor \cdot \rfloor : r \in \mathbb{R} \mapsto \max\{a \in \mathbb{F} \mid a \leq r\} \\ \lceil \cdot \rceil : r \in \mathbb{R} \mapsto \min\{a \in \mathbb{F} \mid a \geq r\} \end{array} \tag{2}$$

The *precision* of an interval $\boldsymbol{x}$ is the number $\lceil \overline{\boldsymbol{x}} - \underline{\boldsymbol{x}} \rceil$. An n-ary box $\mathbf{x}$ is a Cartesian product of intervals $\boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n$. The precision of a box is defined as the precision of its largest component. The change from real to interval computations consists in replacing every set of real numbers by an enclosing interval. More precisely, the convex hull, $\mathsf{hull}\,(\rho)$, of a set $\rho \subset \mathbb{R}$ is the smallest interval enclosing ρ with respect to set inclusion. A computational definition of the hull is $[\lfloor \inf(\rho) \rfloor, \lceil \sup(\rho) \rceil]$.

Now the precision test and the splitting step of Algorithm BranchAndPrune can be clarified. The precision test checks out the precision of the current box $\mathbf{y}$. The test succeeds if the precision of $\mathbf{y}$ is smaller than or equal to the number ε. The splitting step divides the current box along one dimension. The selected dimension corresponds to an integer $i \in \{1, \ldots, n\}$ such that the i-th component of $\mathbf{y}$ is less precise than ε. Then $\boldsymbol{y}_i$ is split, which results in the set of intervals $\{\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_k\}$. For reliability issues, the union of those intervals is equal to $\boldsymbol{y}_i$. The splitting method then returns the set of boxes

$$\{\boldsymbol{y}_1 \times \cdots \times \boldsymbol{y}_{i-1} \times \boldsymbol{y}'_j \times \boldsymbol{y}_{i+1} \times \cdots \times \boldsymbol{y}_n\}_{j=1,\ldots,k}.$$

In general, the chosen dimension corresponds to the largest interval and the box is split in two parts (*i.e.*, $k = 2$). However, we will see in the following that these parameters can be modified in `RealPaver` models.

2.2 Interval Arithmetic

Interval arithmetic is a set theoretic extension of real arithmetic. Given an operation $\diamond \in \{+, -, \times, /\}$ and an elementary function $\phi : \mathbb{R} \to \mathbb{R}$, the following relations must hold for every couple of intervals $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{I}$:

$$\begin{aligned} \boldsymbol{x} \diamond \boldsymbol{y} &= \mathsf{hull}(\{x \diamond y \mid (x, y) \in \boldsymbol{x} \times \boldsymbol{y}\}) \\ \phi(\boldsymbol{x}) &= \mathsf{hull}(\{\phi(x) \mid x \in \boldsymbol{x}\}) \end{aligned} \tag{3}$$

These operations are implemented by floating-point computations over interval bounds according to monotonicity properties. For instance, the sum $[a, b]+[c, d]$ is equal to $[\lfloor a + c \rfloor, \lceil b + d \rceil]$. Interval reasoning can be extended to complex functions using the so-called interval evaluation method. Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and a box $\mathbf{x}$, each real number in f is replaced with its convex hull, each variable is replaced with its domain in $\mathbf{x}$, and each operation is replaced with the corresponding interval operation. This interval expression is evaluated using interval arithmetic. Let $\boldsymbol{f}(\mathbf{x})$ denote the resulting interval. The fundamental property of interval arithmetic [Moore 1966] is that $\boldsymbol{f}(\mathbf{x})$ is a superset of the range of f over $\mathbf{x}$.

Example 1. Consider the function $f(x_1, x_2) = x_1^2 - x_2 + 6$ and the box $\mathbf{x} = [1, 2] \times [-1, 1]$. Then the range of f over $\mathbf{x}$ is included in the interval

$$\boldsymbol{f}(\mathbf{x}) = [1, 2]^2 - [-1, 1] + [6, 6] = [1, 4] + [5, 7] = [6, 11].$$

Given a constraint c, interval evaluation can be used to check whether a box is not feasible with respect to c (violation), or whether a box is included in the solution set of c (certain satisfaction). First, it is worth noticing that some constraint $f \bowtie 0$ is equivalent to $f \in I$ such that $I = [0, 0]$ for equations, and $I = [0, +\infty]$ or $I = [-\infty, 0]$ for inequality constraints. Then, given a box $\mathbf{x}$ and a constraint $c : f \in I$, we have the following interval tests:

$$\begin{array}{lll} \textit{i.} & c \text{ is violated by } \mathbf{x} \text{ if} & \boldsymbol{f}(\mathbf{x}) \cap I = \varnothing \\ \textit{ii.} & c \text{ is certainly satisfied by } \mathbf{x} \text{ if} & \boldsymbol{f}(\mathbf{x}) \cap I = \boldsymbol{f}(\mathbf{x}) \end{array} \tag{4}$$

Now the inner test of Alg. BranchAndPrune can be formally defined. The condition succeeds if all the constraints from $\mathcal{C}$ are shown to be certainly satisfied by the box $\mathbf{y}$. In this case, the box is added to the inner approximation.

These interval tests can be used as a pruning procedure in the branch-and-prune algorithm. However, this strategy does not solve hard problems in general. The main reason is that an unfeasible region of the search space needs to be isolated before its elimination. There is a need for early pruning by means of reduction algorithms. Two types of reduction techniques are implemented in `RealPaver`, the interval Newton method (see Section 2.3) and consistency techniques (see Section 2.5).

2.3 Interval Analysis

To address reliability issues in the numerical processing of nonlinear systems, a number of methods from numerical analysis have been adapted to deal with interval arithmetic [Moore 1966; Neumaier 1990; Jaulin et al. 2001]. In our framework, these methods are used in the pruning step of the branch-and-prune algorithm.

For instance, the interval Newton method processes a square system of equations $\mathrm{f}(\mathrm{x}) = 0$ in two steps, provided that f is differentiable. Given a starting point $\mathrm{x}^{(0)}$, a first-order Taylor expansion of the system leads to the equation

$$\exists \xi \; \mathrm{f}(\mathrm{x}^{(0)}) + \nabla \mathrm{f}(\xi) \times (\mathrm{x} - \mathrm{x}^{(0)}) = 0 \tag{5}$$

where ξ lies strictly between $\mathrm{x}^{(0)}$ and x. Now, the interval reasoning consists in bounding the derivative in the error term of the Taylor formula by an interval evaluation of the Jacobian matrix $\nabla \mathrm{f}$ over the whole domain. Given $\mathbf{y}$ a box enclosing x and $\mathrm{x}^{(0)}$, we deduce an inclusion relation from (5), as follows:

$$\mathrm{f}(\mathrm{x}^{(0)}) + \nabla \mathrm{f}(\mathbf{y}) \times (\mathrm{x} - \mathrm{x}^{(0)}) \ni 0 \tag{6}$$

In practice, Vector $\mathbf{y}$ corresponds to the variable domains and $\mathrm{x}^{(0)}$ is generally the midpoint of $\mathbf{y}$. Then a sequence of intervals is computed by the interval Gauss-Seidel method for the linear system

$$P \times \nabla \mathrm{f}(\mathbf{y}) \times \mathbf{z} = -P \times \mathrm{f}(\mathrm{x}^{(0)}) \tag{7}$$

where $\mathbf{z}$ is equal to $(\mathrm{x} - \mathrm{x}^{(0)})$, and P is a preconditioning matrix. The new box $N(\mathrm{f}, \mathbf{y}, \mathrm{x}^{(0)})$ is obtained as $\mathrm{x}^{(0)} + \mathbf{z}$. Two properties are guaranteed: The sequence converges and each box from the sequence encloses the solution set.

Systems of inequality constraints may be processed by the Simplex algorithm as described in Yamamura et al. [1998]. Each nonlinear term is replaced with a variable lying in its interval evaluation. Then the Simplex algorithm derives an enclosure of the solution set.

2.4 Existence and Uniqueness Proofs

In some cases, interval methods provide proofs of existence and uniqueness of solutions of nonlinear systems [Kearfott 1996]. The main results originate from the fixed-point theory.

For instance, consider the multivariate interval Newton method previously described. The following properties hold:

—$N(\mathrm{f}, \mathbf{y}, \mathrm{x}^{(0)})$ contains all solutions of the system $\mathrm{f} = 0$ in $\mathbf{y}$. Hence, if the box is empty, then there is no solution.

—if $N(\mathrm{f}, \mathbf{y}, \mathrm{x}^{(0)}) \subset \mathsf{int}(\mathbf{y})$, where $\mathsf{int}(\mathbf{y})$ stands for the interior of $\mathbf{y}$, then there exists a unique solution of the system $\mathrm{f} = 0$ in $\mathbf{y}$.

These algorithms are used in `RealPaver`.

2.5 Constraint Satisfaction Techniques

Constraint satisfaction techniques are algorithms of polynomial-time worst case complexity that implement local reasonings on constraints to remove

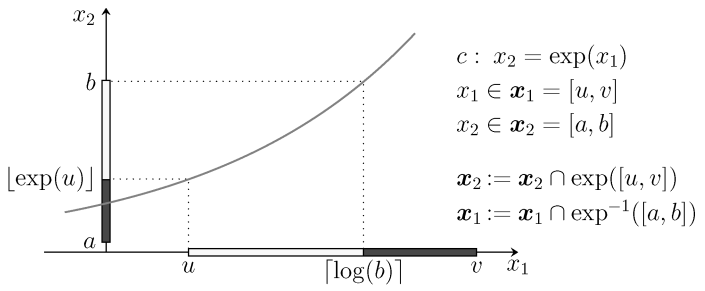
x_2
b
$\lfloor\exp(u)\rfloor$
a
u
$\lceil\log(b)\rceil$
v
x_1
$c:\ x_2 = \exp(x_1)$
$x_1 \in \boldsymbol{x}_1 = [u, v]$
$x_2 \in \boldsymbol{x}_2 = [a, b]$
$\boldsymbol{x}_2 := \boldsymbol{x}_2 \cap \exp([u, v])$
$\boldsymbol{x}_1 := \boldsymbol{x}_1 \cap \exp^{-1}([a, b])$

Fig. 1. Inversion of the constraint $x_2 = \exp(x_1)$ for computing domain reductions.

inconsistent values from variable domains. In our framework, they are enforced in the pruning step of the branch-and-prune algorithm. This is a general approach to solving nonlinear systems since functions do not need to be differentiable and systems can be heterogeneous and nonsquare. Large and sparse nonlinear systems from automatic control [Jaulin et al. 2001] have been successfully solved (from 10^4 to 10^5 constraints).

Local domain pruning may be obtained by considering constraint projections. Given a constraint $c(x_1, \ldots, x_n)$ and an integer $i \in \{1, \ldots, n\}$, define the i-th projection of c as the set

$$\pi_i(c) = \{a_i \in \mathbb{R} \mid \exists (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, n) \in \mathbb{R}^{n-1} : \ c(a_1, \ldots, a_n)\}. \qquad (8)$$

Given a box $\mathbf{x}$, values from the interval $\boldsymbol{x}_i$ that do not belong to the i-th projection of c can be removed while preserving the solution set of the constraint system. Unfortunately, the general problem of determining the set of inconsistent values is uncomputable over the real numbers. In practice, a subset of the set of inconsistent values is computed by means of interval reasonings. It is guaranteed that computed approximations of projections are reliable. Two techniques closely related to hull-consistency and box-consistency are implemented in `RealPaver`. These techniques are described below, and heuristics as well as solving strategies will be described in Section 4.2.

The constraint inversion procedure uses the inverse real operations as illustrated in Figure 1. The initial box $[u, v] \times [a, b]$ is reduced to $[u, \lceil\log(b)\rceil] \times [\lfloor\exp(u)\rfloor, b]$ by means of interval arithmetic. The resulting box is said to be *hull-consistent*, since every resulting domain is equal to the hull of the corresponding constraint projection. A numerical constraint inversion algorithm called `HC4revise` for processing complex constraints has been introduced in Benhamou et al. [1999] and defined as a chain rule in Granvilliers and Benhamou [2001]. This is a direct method that is optimal if variables occur once in constraints.

The dichotomous search procedure combines the interval tests of Equation (4) with a splitting process over one variable domain. Interval tests are used to eliminate subdomains that violate the considered constraint. The resulting domain is the enclosure of the subdomains that have not been discarded. This domain is said to be *box-consistent*, since it cannot be reduced using the tests. The case for an equation is shown in Figure 2. All the external white rectangles can be rejected since they do not cross the axis. Given as input the domain $[a, b]$, the final domain is $[c, d]$. This technique has been introduced

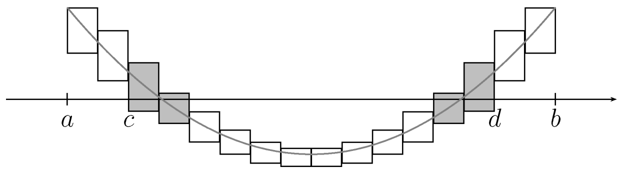

Fig. 2. Search and interval tests over the equation $f(x) = 0$.

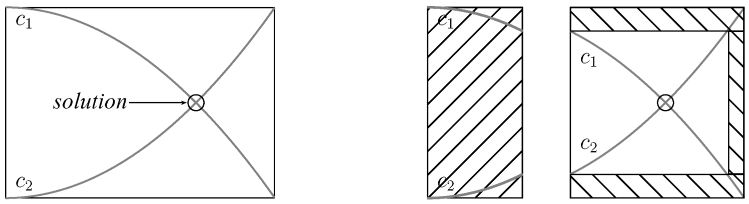

Fig. 3. Locality problem and strong consistency technique.

in Benhamou et al. [1994] and called `BC3revise` in Benhamou et al. [1999]. This iterative method has been shown to be more efficient than constraint inversion if variables occur more than once in constraints.

Given a constraint system, constraint projections have to be processed in sequence, in order to reach the consistency of the whole problem. The fixed-point algorithm implementing such a sequence of computations is called constraint propagation [Cousot and Cousot 1977; Apt 2000]. The algorithm terminates in finite time, since every step is contracting and the computational domain is finite.

Example 2. Consider the system $(\{x_2 = x_1^2, x_1^2 + x_2^2 = 2\}, [-10, 10]^2)$ whose solutions are $(1, 1)$ and $(-1, 1)$. The first constraint leads to reduce the domain of x_2 to $[0, 10]$ (x_2 must be positive). The second constraint allows one to contract the domain of x_1 to $[\lfloor -\sqrt{2} \rfloor, \lceil \sqrt{2} \rceil]$, and so on. The result is the box $[-1.19, 1.19] \times [0.76, 1.42]$. Unfortunately, this process does not compute the enclosure $[-1, 1] \times [1, 1]$ of the solution set. This is due to the locality problem.

The locality problem is due to the strategy for reducing domains, since every constraint projection is processed independently. In Figure 3, the system represents an intersection of two curves, c_1 and c_2. In this case, the box cannot be reliably reduced using c_1 because this would lose solutions of c_1 (idem for c_2). This problem has led to the definition of stronger consistency techniques, in particular, kB-consistencies [Lhomme 1993]. The main idea depicted in Figure 3 is to prove the inconsistency of a sub-box using the constraint propagation algorithm. In this case, the leftmost sub-box is discarded and the rightmost sub-box may be contracted. If this procedure is iterated, then a tight enclosure of the solution can be computed. However, this process is computationally expensive [Puget and Van Hentenryck 1998], and it should not be used as a default solving strategy.

We will describe in Section 4 an efficient strategy to combine consistency algorithms and the interval Newton method.

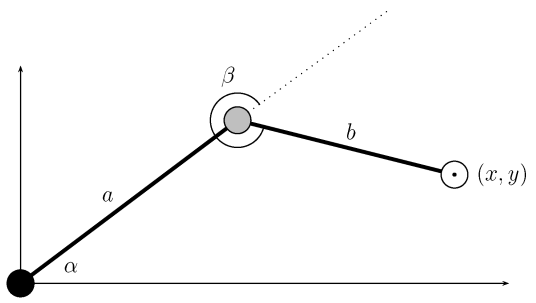

Fig. 4. Kinematics problem with two links and two revolute joints.

3. PROBLEM MODELING

`RealPaver` is a software that takes as input a model in the form of a constraint system and some parameters values, and returns a box approximation of the set of solutions in text format. In the following, we describe the modeling of a set of problems chosen to be representative of the capabilities of the solver.

3.1 Direct and Inverse Problems

Consider the kinematics chain in Figure 4 representing a fixed revolute joint, a moving link of fixed length a, a moving revolute joint, and a moving link of fixed length b. The origin of the 2D coordinate system is the center of the fixed joint. The end of the last link simulates the hand of a robot.

The constraint model (9) defines relations between the angles α, β, and the position of the hand (x, y). Moreover, consistent values for x and y belong to the interval $[-(a+b), a+b]$, and the angles are defined in $[-\pi, \pi]$.

$$\begin{cases} a\sin(\alpha) = b\sin(\beta - \alpha) + y \\ a\cos(\alpha) = x - b\cos(\beta - \alpha) \end{cases} \tag{9}$$

Each revolute joint represents a single rotational degree of freedom. As a consequence, the system is under-constrained. Two variables have to be fixed in order to solve (9). When the angles are fixed, the problem is usually called the direct problem. When the position of the hand is fixed, the problem is said to be inverse. Let us model in `RealPaver` the inverse problem, given $a = 4$, $b = 2$ and $(x, y) = (2.5, 4)$:

```
/* RealPaver model: kinematics problem */
Constants    a = 4, b = 2;

Variables    x in [-(a+b),a+b],  alpha in [-@pi,@pi],
             y in [-(a+b),a+b],  beta  in [-@pi,@pi];

Constraints  a*sin(alpha) = b*sin(beta-alpha) + y,
             a*cos(alpha) = x - b*cos(beta-alpha),
             x = 2.5,  y = 4;
```

The variable domains are bounded. The extreme values for the position of the hand correspond to aligned axes. The predefined constant `@pi` stands for π. Note that the set of predefined constants are described in the user's manual [Granvilliers 2003]. Two boxes are computed, each one containing a solution. The first box returned is given hereafter:

```
OUTER BOX 1  [precision: 4.88e-15, elapsed time: 0 ms]
  x     = 2.5
  y     = 4
  alpha in [1.445406365667148 , 1.445406365667151]
  beta  in [1.429703667339 , 1.429703667339005]
```

There is a symmetry of the position of the second joint with respect to the axis between the origin and the position of the hand. The symmetry can be broken using an additional constraint, for instance, $a\cos(\alpha) \leq x$. In this case, only the first solution is derived.

3.2 Proof of Inconsistency

A model can be proved to be inconsistent if no box is computed. This property may be very useful, in particular, for critical applications. For instance, given the previously described kinematics problem, suppose that the hand must not enter a surface delimited by a circle of center (6, 8) and of radius 1. It suffices to add the constraint

$$(x-6)^2 + (y-8)^2 \leq 1$$

in the model (where x and y are not fixed) and to run `RealPaver`. The following result guarantees that for all values of x and y, the system is inconsistent.

```
END OF SOLVING
  Property:     no solution in the initial box
  Elapsed time: 0 ms
```

3.3 Mixed Problem

`RealPaver` is able to process mixed integer nonlinear systems. Consider a dynamical system modeled by a decreasing exponential term

$$y(t) = 2\exp(-0.5t).$$

Time is represented by variable t, which is a discrete variable, while the output of the system $y(t)$ is continuous. Consider the problem of finding t such that $y(t)$ is smaller than 0.025. The model follows where t is constrained to be an integer variable.

```
/* RealPaver model: mixed problem */
Variables   real y in [0,10],
            int  t in [0,10];

Constraints y = 2*exp(-0.5*t),
            y <= 0.025;
```

This problem has two solutions which are easily computed by `RealPaver`. As suggested in Benhamou and Older [1997], integer variables can be handled as real variables such that each domain modification is followed by truncation of bounds to the nearest integers inside the interval domain.

```
OUTER BOX 1 [precision: 8.88e-16, elapsed time: 0 ms]
  y in [0.02221799307648417 , 0.02221799307648506]
  t = 9

OUTER BOX 2  [precision: 8.88e-16, elapsed time: 0 ms]
  y in [0.01347589399817049 , 0.01347589399817138]
  t = 10
```

3.4 Disjunctive Constraints and Domains

Nonlinear systems are processed as conjunctions of constraints. In this framework, disjunctive constraints or domains "with holes" can be modeled using the *minimum* function. Note that the following preprocessing operations have to be done by hand. Consider the problem of two variables x and y such that either y is negative or y is greater than x. The disjunction of constraints is rewritten as

$$y \leq 0 \ \vee \ x - y \leq 0.$$

In other words, at least one of both expressions must be negative (*i.e.*, their minimum must be negative). The corresponding constraint is

$$\min(y, x - y) \leq 0.$$

If variable x belongs to the union of intervals $[-10, -2] \cup [3, +\infty]$, then it suffices to define x in the interval $[-10, +\infty]$ and to add the constraint

$$\min(x + 2, 3 - x) \leq 0.$$

Domains with more than two holes can be handled in the same way, using minimum and maximum operations. For instance, the domain $x \in [-10, -2] \cup [0, 1] \cup [3, +\infty]$ is implemented by

$$\min(x + 2, \min(\max(-x, x - 1), 3 - x)) \leq 0.$$

Note that these operations are not differentiable. Fortunately, consistency techniques just need that operations can be evaluated over intervals.

3.5 Approximation of a Continuum of Solutions

Consider the problem of approximating the surface defined by

$$y \geq x^2 - 1 \quad (x, y) \in [-2, 2] \times [-1.5, 1],$$

given a precision of 10^{-2}. Given a domain splitting strategy in two parts, `RealPaver` computes 130 boxes, the union of which encloses the solution set. Figure 5 shows the approximation of the surface, where the light-coloured boxes are included in the solution set, and black boxes enclose the frontier. It is worth

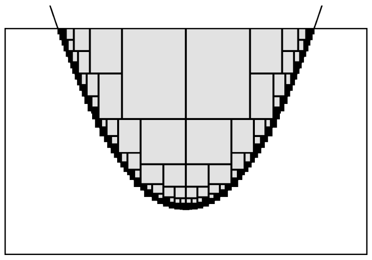
Fig. 5. Approximation of a continuum of solutions.

noticing that such approximations can be represented by 2^k-trees [Sam Haroud and Faltings 1996]. Efficient computational techniques based on computational geometry algorithms have been introduced in Vu et al. [2002].

4. STRATEGY ENGINE

The branch-and-prune algorithm uses several independent procedures: Reduction algorithm for boxes, choice of variable to split, precision of output boxes, etc. In the following, we discuss these elements of strategy and show how they can be tuned using the modeling language.

4.1 Splitting

`RealPaver` is able to tackle the following problems, given a natural number n:

(1) The computation of (at most) n boxes at the desired precision. This mode is classically used for approximating discrete sets of solutions. Boxes are processed in depth-first order (last-in-first-out strategy), which allows one to derive boxes at the given precision as fast as possible. This makes sense if the motivation is to obtain only one solution for problems having large numbers of solutions. However, completeness is not guaranteed since some solutions may be lost.

(2) The computation of (at most) n boxes that contain all the solutions of the model. Such an approximation is called a paving. This mode is useful for representing continuous sets of solutions by a finite number of boxes. Boxes are processed in breadth-first order (first-in-first-out strategy), which allows one to derive boxes having comparable precisions.

Moreover, a third mode is available which consists in disabling the splitting process. In this case, only a reduction phase is performed. Optional flags can be added in models to tune the splitting strategy. For instance, the following piece of code corresponds to the computation of one box at precision 10^{-4}, with a splitting of domains in two parts.

```
Split precision = 1.0e-4        ,
      mode      = points        ,     /* other mode: paving */
      parts     = 2             ,
      number    = 1             ,
      choice    = largest_first ;
```

Splitting of a box consists in choosing and subdividing one domain. The variable choice strategy has been shown to greatly influence efficiency. As in `Numerica`, two strategies are implemented in `RealPaver`: `largest_first`, as shown above, such that the largest domain is selected; and `round_robin`, which alternates variable choices according to a fixed ordering, more precisely, the ordering in which variables are declared in models. The default strategy is `round_robin`.

Finally, it is also possible to time out the computation by adding the following line in models.

```
Time = 1000 ;   /* maximum computation time in milliseconds */
```

4.2 Consistency Techniques

The generation of constraint propagation algorithms from heterogeneous nonlinear systems is a piece of strategy that can be modified in the generic branch-and-prune approach. The default strategy is as follows. For each constraint containing at least one variable occurring once, the inversion procedure is used. For each constraint and for each variable occurring more than once in the constraint, the dichotomous search procedure is implemented. Finally, the interval Newton method is enforced over a square system of equations, if such a system can be extracted from the initial system. This strategy has been shown in Granvilliers [2001] to be the most efficient in average over a standard set of benchmarks for nonlinear systems, and it should be used most of the time. However, reduction algorithms may be tuned for specific problems. For instance, the following piece of code implements a 3B-consistency at precision 10^{-4} using the inversion procedure (`hc4`) as a reduction technique.

```
Consistency  strong = 3B     ,  /* strong consistency    */
             width  = 1.0e-4 ,  /* width of boxes for 3B */
             local  = hc4    ;  /* reduction technique   */
```

An adaptation of 3B-consistency using box-consistency as the main reduction procedure has been shown to be very efficient for solving a well-known transistor problem in Puget and Van Hentenryck [1998]. A new algorithm based on `HC4revise` has been developed in `RealPaver` and the results have been reported in Granvilliers and Benhamou [2001]. The tuning section of the model is presented below. The `Output` section specifies that each floating-point number is printed with 16 significant digits, and that each interval is presented as the midpoint and the error bounds. The `Split` section states that the precision of the branch-and-prune algorithm is 10^{-12}, that the variable to be split is the one having the largest domain, and that each domain is split in three parts. Finally, the `Consistency` section states that 3B-consistency at precision 10^{-4} is used as the pruning technique, and that the precision of the dichotomous search algorithm is 0 (*i.e.*, the maximum precision).

```
Output      digits    = 16        ,
            style     = midpoint  ;

Split       precision = 1.0e-12    ,
            choice = largest_first ,
            parts  = 3             ;

Consistency strong = 3B     ,
            width  = 1.0e-4 ,
            phi    = 0      ;
```

Given this tuning, only one box is derived.

```
OUTER BOX 1  [precision: 9.24e-13, elapsed time: 35,430 ms]
  X[1] = 0.8999999526168566 + [-4.441e-16,+5.551e-16]
  X[2] = 0.4499874719815348 + [-4.446e-14,+4.446e-14]
  X[3] = 1.000006482465264 + [-9.259e-14,+9.281e-14]
  X[4] = 2.000068541624237 + [-1.177e-13,+1.181e-13]
  X[5] = 7.999971440508141 + [-2.007e-13,+2.016e-13]
  X[6] = 7.999692684217048 + [-4.619e-13,+4.619e-13]
  X[7] = 5.000031275930076 + [-1.794e-13,+1.794e-13]
  X[8] = 0.9999877234567925 + [-1.865e-14,+1.865e-14]
  X[9] = 2.000052483486345 + [-1.092e-13,+1.092e-13]
```

5. PACKAGE AND DISTRIBUTION

The code is organized as a set of modules written in the C language. The whole package contains 13000 lines of C code.

(1) The parsing module is composed of regular expressions for lexical units and a grammar of the modeling language, which is given in the user's manual. The analyzers are generated by the GNU tools `flex` and `bison` [Free Software Foundation 2002].
(2) The constraint module concerns the internal representation of constants, variables, functional expressions, and constraints. A functional expression is represented by a tree where each node is either an operation node, a variable node, or an interval node. A constraint is given by two functions and a relation symbol.
(3) The interval module implements the interval type, the interval union type, interval arithmetic, interval evaluation, and differentiation in backward mode. An interval is a couple of IEEE floating-point numbers of type `double`.
(4) The pruning module contains consistency algorithms, a matrix library, and the interval Newton method. In particular, there are the inverse interval operations, the inversion algorithm, the dichotomous search algorithm, preconditioning techniques, and the interval Gauss-Seidel method.
(5) The search module implements the branch-and-prune algorithm and different splitting strategies (variable choices, splitting method).

The main function is responsible for creating global data structures for constraint system representations, calling the parser, applying the strategy engine for generating solving algorithms, calling the solver, and managing help and profiling information.

RealPaver is freely available for academic and research purposes. The full sources, a documentation, and executable codes for ix86 computers under Linux and Cygwin, Sun Sparc computers under Solaris 8, and SGI computers under IRIX 6.5 can be freely downloaded from the Internet. The compilation of RealPaver requires an ANSI C compiler like `gcc 3.2`, the compiling tool `GNU make`, and the parsing tools `GNU flex/bison`.

The compressed tarball file has to be copied in a directory where write permissions are available. The installation is then done using the following command lines that implement uncompression, configuration, compilation, and linkage for version 0.3 of the software.

```
gzip -c -d realpaver-0.3.tar.gz | tar xf -
cd realpaver-0.3
./configure
gmake       # or make if gmake is not available
```

The running command line is as follows, where `filename` is the file name of the source model.

```
./realpaver filename
```

Two execution modes are available, the default one and the profiling mode, which can be enabled at configuration time as follows.

```
./configure --enable-profile
```

Profiling information concerns the constraint system, timings for the different algorithms, and the number of evaluated interval operations. For instance, given the previously introduced transistor problem, the following information is printed. The density indicates whether constraints contain a few variables (sparse systems) or not (dense systems). Values range from n^{-1} for sparse systems to 1 for dense systems.

```
SOLVING
  System: 12 x 12 of equations, density: 0.368
  Reduction: 3B(0.0001) / HC4 + Newton
  Split: points, largest-first, 3 parts, precision: 1e-12
```

The following information is printed after solving. More than six thousands of splitting steps have been computed. The timings show that most of the computation time is spent in the inversion procedure (HC4).

```
PROFILING
  Split: 6,254 / 18,763 boxes examined
  Solving time in consistency algorithms:
     in BC3:         0 ms
     in HC3:         0 ms
     in HC4:    58,971 ms
     in Newton: 17,079 ms
```

The numbers of evaluated interval operations are also given. In particular, it can be noticed that the logarithm is used to inverse the exponential terms, while this operation is not used in the problem expression.

```
  Number of interval operations:
             add:  53,630,709
             sub:  35,268,246
             mul:  80,194,514
             div:  48,438,791
            ediv:  48,438,790
             exp:   3,979,170
             log:   3,105,492
                  -----------
           total: 224,616,922
```

6. CONCLUSION AND FUTURE DEVELOPMENTS

RealPaver is a modeling language embedding advanced interval-based constraint solving techniques for nonlinear systems. Practical evidence of the good performance of the solver has been given on a large set of applications. Moreover, the language is simple enough to allow nonexperts in constraint programming to easily model constraint systems.

A short-term research direction is to improve the computation of inner boxes, and to implement a strategy which is able to automatically adapt the consistency level (2B or 3B) to the input problems. We also plan to support known exchange formats and to define an API for integration of functionalities in other softwares. An executable code for Windows will be available.

In the future, we plan to integrate new solving techniques. Symbolic computations have been shown to greatly improve interval computations, particularly for interval evaluation [Ceberio and Granvilliers 2002a] and consistency techniques [Ceberio and Granvilliers 2002b]. A symbolic preprocessing can be implemented before numerical solving while preserving the software architecture. The use of the Simplex algorithm for linear relaxations is also a powerful approach [Yamamura et al. 1998; Lebbah et al. 2002]. As future long-term research, we plan to extend RealPaver to new classes of problems, such as constrained optimization, differential equations, and quantified constraints.

ACKNOWLEDGMENTS

The authors are grateful to Frédéric Goualard who has greatly contributed to the study of the interval-based constraint satisfaction techniques implemented in RealPaver. Luc Jaulin, Vladik Kreinovich, Jean-Pierre Merlet, Michel Rueher, Alexander Semenov, and Pascal Van Hentenryck are gratefully acknowledged for their useful comments on a preliminary version of this article.

REFERENCES

ALEFELD, G. AND HERZBERGER, J. 1983. *Introduction to Interval Computations*. Academic Press, NY.

APT, K. R. 2000. The role of commutativity in constraint propagation algorithms. *ACM Trans. Program. Lang. Syst. 22,* 6, 1002–1036.

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)*. The MIT Press, Las Cruces, Calif., 230–244.

BENHAMOU, F., MCALLESTER, D., AND VAN HENTENRYCK, P. 1994. CLP(intervals) revisited. In *Proceedings of the Logic Programming Symposium (ILPS)*. MIT Press, Ithaca, N.Y. 124–138.

BENHAMOU, F. AND OLDER, W. J. 1997. Applying interval arithmetic to real, integer and boolean constraints. *J. Logic Program. 32,* 1, 1–24.

BERLEANT, D. AND KREINOVICH, V. 2003. cs.utep.edu/interval-comp/main.html.

BERZ, M. AND MAKINO, K. 2002. *COSY INFINITY: User's Guide and Reference Manual*. Michigan State University. http://cosy.pa.msu.edu/.

CEBERIO, M. AND GRANVILLIERS, L. 2002a. Horner's rule for interval evaluation revisited. *Comput. 69,* 1, 51–81.

CEBERIO, M. AND GRANVILLIERS, L. 2002b. Solving nonlinear equations by abstraction, gaussian elimination, and interval methods. In *Proceedings of the International Workshop on Frontiers of Combining Systems*. LNCS, vol. 2309. Springer, Berlin, 117–131.

COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, R. Sethi, ed. ACM Press, Los Angeles, Calif. 238–252.

FOURER, R., GAY, D. M., AND KERNIGHAN, B. W. 2003. *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed. Duxbury Press, Pacific Grove, CA.

Free Software Foundation 2002. *Bison 1.35*. Free Software Foundation.

GRANVILLIERS, L. 2001. On the combination of interval constraint solvers. *Reliable Comput. 7,* 6, 467–483.

GRANVILLIERS, L. 2003. *RealPaver User's Manual, version 0.3*. University of Nantes, France. www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver.

GRANVILLIERS, L. AND BENHAMOU, F. 2001. Progress in the solving of a circuit design problem. *J. Global Optimization 20,* 2, 155–168.

GRANVILLIERS, L., BENHAMOU, F., AND HUENS, E. 2001. Nonlinear constraint propagation. Deliverable of the COCONUT Project.

IEEE. 1985. *IEEE Standard for Binary Floating-Point Arithmetic*. IEEE Computer Society. Reaffirmed 1990.

ILOG 2001. *Ilog Solver 5.2*. ILOG.

JAULIN, L., KIEFFER, M., DIDRIT, O., AND WALTER, E. 2001. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, Berlin.

KEARFOTT, R. B. 1996. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dordrecht, Netherlands.

LEBBAH, Y., RUEHER, M., AND MICHEL, C. 2002. A global filtering algorithm for handling systems of quadratic equations and inequations. In *Proceedings of the Principles and Practice of Constraint Programming (CP)*. LNCS, vol. 2470. Springer, Berlin, 109–123.

LHOMME, O. 1993. Consistency techniques for numeric CSPs. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI)*. Morgan Kaufman, ed. Chambéry, France, 232–238.

MEIER, M. AND SCHIMPF, J. 1993. ECLiPSe User Manual. Tech. Rep. ECRC-93-6, ECRC (European Computer-Industry Research Centre), Munich, Germany.

MERLET, J.-P. 2002. *ALIAS: An Algorithms Library of Interval Analysis for Equation Systems*. INRIA. http://www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html.

Moore, R. E. 1966. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J.

Neumaier, A. 1990. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, UK.

O'Sullivan, B. 1999. Constraint-Aided conceptual design. Ph.D. thesis, University College Cork.

PrologIA. 1996. *Prolog IV: Constraints inside*. Parc Technologique de Luminy—Case 919, 13288 Marseille cedex 09, France. Prolog IV reference manual.

Puget, J. and Van Hentenryck, P. 1998. A constraint satisfaction approach to a circuit design problem. *J. Global Optimization 13,* 1, 75–93.

Sam Haroud, D. and Faltings, B. 1996. Consistency techniques for continuous constraints. *Constraints 1*, 85–118.

Schittkowski, K. 2002. *Numerical Data Fitting in Dynamical Systems*. Kluwer Academic Publishers, Dordrecht, Netherlands.

Semenov, A., Kashevarova, T., Leshchenko, A., and Petunin, D. 1997. Combining various techniques with the algorithm of subdefinite calculations. In *Proceedings of the International Conference on the Practical Application of Constraint Technology*. LNCS, vol. 1277. Springer, Berlin, 287–306.

Stewart, D. 1965. A platform with 6 degrees of freedom. In *Proceedings of the Institution of Mechanical Engineers 180,* 15, 371–386.

Van Hentenryck, P., McAllester, D., and Kapur, D. 1997. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal. 34,* 2, 797–827.

Van Hentenryck, P., Michel, L., and Deville, Y. 1997. *Numerica: A Modeling Language for Global Optimization*. MIT Press, Cambridge, Mass.

Vu, X.-H., Sam-Haroud, D., and Faltings, B. 2002. Approximation techniques for non-linear problems with continuum of solutions. In *Proceedings of the Abstraction, Reformulation and Approximation Conference*. LNCS, vol. 2371. Springer, Berlin, 224–241.

Yamamura, K., Kawata, H., and Tokue, A. 1998. Interval analysis using linear programming. *BIT 38*, 188–201.

Received November 2003; revised June 2005; accepted July 2005