Journal of Global Optimization **20:** 159–172, 2001.
© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

Progress in the Solving of a Circuit Design Problem

LAURENT GRANVILLIERS and FRÉDÉRIC BENHAMOU
IRIN, Université de Nantes, B.P. 92208, F-44322 Nantes Cedex 3, France (e-mail: granvilliers,benhamou@irin.univ-nantes.fr)

(Received 8 March 2000; accepted in revised form 4 February 2001)

Abstract. This paper describes a new global branch-and-prune algorithm dedicated to the solving of nonlinear systems. The pruning technique combines a multidimensional interval Newton method with HC4, a state of the art constraint satisfaction algorithm recently proposed by the authors. From an algorithmic point of view, the main contributions of this paper are the design of a fine-grained interaction between both algorithms which avoids some unnecessary computation and the description of HC4 in terms of a chain rule for constraint projections. Our algorithm is experimentally compared, on a particular circuit design problem proposed by Ebers and Moll in 1954, with two global methods proposed in the last ten years by Ratschek and Rokne and by Puget and Van Hentenryck. This comparison shows an improvement factor of five with respect to the fastest of these previous implementations on the same machine.

Key words: Constraint satisfaction, Interval analysis, Newton method, Automatic differentiation, Circuit design.

1. Introduction and Related Work

To address soundness and completeness issues in the numerical processing of nonlinear systems, a number of methods based on interval analysis and adapted from numerical analysis have been developed in the last thirty years. One may cite Taylor expansions, Newton methods and Gauss-Seidel iterations. On the other hand, in the last decade, some authors (Cleary [5], Hyvönen [10], Older [14], Benhamou [3], Lhomme [11], Faltings [7], Van Hentenryck [17]) have shown that constraint satisfaction techniques can be a powerful enhancement to traditional interval methods [12, 13]. One of the most efficient solvers applying these ideas, Numerica [18], is based on a combination of local consistency (domain pruning) and bisection-based search. The core algorithm of Numerica is a branch-and-prune algorithm [17]) where the pruning step combines fixed-point based domain pruning algorithms with a multidimensional interval Newton method. For instance, this algorithm was used in [15] to efficiently solve the circuit design problem from Ebers and Moll [6].

In this paper, we show that the constraint satisfaction algorithm HC4 [1] implements a chain rule for constraint projections which is very similar to the chain rule for derivatives computed in backward mode [8]. We then take advantage of this similarity to combine automatic differentiation and domain pruning in a single algorithm. Given a set of trees representing function expressions, the computation of their interval evaluations is achieved by a bottom-up traversal. The derivatives and constraint projections are then computed by a top-down traversal of these trees, applying elementary derivation or projection rules. Finally, a new pruning algorithm is designed, combining HC4 with a multidimensional interval Newton method. This algorithm has two main specific advantages. On the one hand, the interval evaluations performed in HC4 are reused for evaluating the Jacobian. On the other hand, possible slow convergence behaviors are avoided by replacing fixed-point iterations with a single tree traversal, while preserving the final output properties, and a global fixed-point is no more computed. As a result, more bisections are performed but the pruning cost is lower.

The second contribution of this paper concerns the experimental results achieved by this algorithm on a classical benchmark. Ebers and Moll's circuit design problem [6] is considered as a challenging benchmark for local and global methods. In the last years, two interval-based techniques were successfully applied: Ratschek and Rokne's algorithm [16] has derived the solution with a precision of four significant digits in (approximatively) fourteen months using a network of thirty Sun Sparc 1 workstations; Puget and Van Hentenryck's method [15] has solved the problem with a precision of eight significant digits in about forty minutes using a Sun Sparc UltraII workstation. Our algorithm computes the solution with a precision of twelve significant digits in four minutes (time to check global optimality is eight minutes) using a Sun Sparc UltraII workstation. The result comes from the average improvement of HC4 with respect to the classical box consistency algorithm for problems with single occurrences of variables in constraints (as shown in [1]) and from the fine-grained cooperation between the implemented pruning techniques.

The rest of the paper is organized as follows: Section 2 introduces some material from interval arithmetic and local consistency techniques. Section 3 presents the chain rule for constraint projections. Section 4 presents the new pruning algorithm. Experimental results are discussed in Section 5. Directions for future research are sketched in Section 6.

2. Preliminaries

Let $\mathbb{R}$ denote the set of real numbers. A *floating-point interval* is a connected set of reals bounded by floating-point numbers. The notation $[a, b]$ is used as a shorthand for the set $\{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$. Let $\mathbb{I}$ be the set of closed intervals. The *width* of $[a, b]$ is the quantity $b - a$ rounded towards $+\infty$. Let $\mathsf{Hull}(\rho)$ be the smallest interval containing any relation $\rho \subseteq \mathbb{R}$, called the *hull* of ρ. A relation $\rho \subseteq \mathbb{R}$ is said to be *interval convex* if $\rho \cap I \in \mathbb{I}$ for every $I \in \mathbb{I}$. The same symbols are used hereinafter to denote operations on $\mathbb{R}$ and $\mathbb{I}$, assuming there is no possible confusion.

A *term* is an atomic formula built from $\mathbb{R}$, a set of operations and a set of real-valued variables. To every variable x is associated a domain $X \in \mathbb{I}$ verifying $x \in X$. Let $\boldsymbol{X}$ denote the Cartesian product of all variable domains, called a *box*. A *constraint* is an atomic formula built from the set of terms and the set of binary relation symbols $\{=, \leqslant, \geqslant\}$. Let ρ_c be the relation associated to any constraint c (that is the set of elements verifying c under the usual interpretation of function and relation symbols) and $\mathsf{Var}(c)$ the set of variables occurring in c. Two constraints are said to be *equivalent* if they define the same relation.

We define now the notions of *projection* and *cylindrification* of the relation of a constraint (respectively a restriction and an extension of the relation to another set of dimensions).

DEFINITION 1. *Let $c(x, x_1, \dots, x_k)$ be a constraint. The projection of c over x is the relation*

$$\{a \in \mathbb{R} \mid \exists (a_1, \dots, a_k) \in \mathbb{R}^k : (a, a_1, \dots, a_k) \in \rho_c \cap \boldsymbol{X}\}.$$

DEFINITION 2. *Let $c(x_1, \dots, x_k)$ be a constraint. The cylindrification of c over $\mathbb{R}^n$, $k < n$, is the relation*

$$\rho_c \times \underbrace{\mathbb{R} \times \cdots \times \mathbb{R}}_{n-k}.$$

Given two constraints $c(x_1, \dots, x_k)$ and $c'(x_1, \dots, x_n)$, $k < n$, c is said to be *more general* than c' if $\rho_{c'} \subseteq (\rho_c \times \mathbb{R} \times \cdots \times \mathbb{R})$.

The previously-introduced notions are illustrated by Example 1.

EXAMPLE 1. *Let us consider a term $x^2 + y^2 - 1$ over the variables x and y, a constraint $x^2 + y^2 - 1 \leqslant 0$, and a box $[0, 1.5] \times [-1.5, 0]$. The associated relations are described by means of the following figure:*

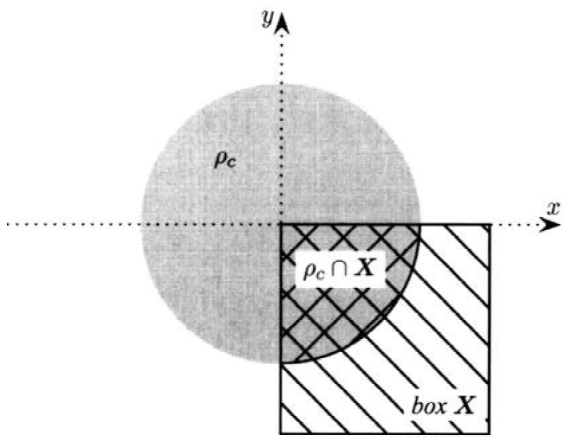


Figure 1.

The projection of c over x corresponds to the intersection of the disk with the x's axis and the box, that is $[0, 1]$. *For instance, the cylindrification of c over $\mathbb{R}^3$ is the ternary relation* $\{(a, b, c) \in \mathbb{R}^3 \mid a^2 + b^2 - 1 \leqslant 0\}$.

A *fresh variable* is a variable of domain $]-\infty, +\infty[$ that does not occur in any constraint. Given a sub-term u occurring in a constraint c (resp. a term f) and a fresh variable z, let $c[u \leftarrow z]$ (resp. $f[u \leftarrow z]$) denote the constraint (resp. the term) obtained from c (resp. f) by replacing u with z. The projection of c over u is then defined as the projection of $c[u \leftarrow z]$ over z where the domain of z is the image of u over the variable domains. Given a term f, u a sub-term of f, and z a fresh variable, $f[u \leftarrow z](a_1, \ldots, a_n, a)$ corresponds to the evaluation of $f[u \leftarrow z]$ for the values $a_1, \ldots, a_n$ of the variables of f and the value a of z; ∂_u^f stands for the derivative of f with respect to u, that is $(\partial f[u \leftarrow z]/\partial z)[z \leftarrow u]$.

A term is represented as a *directed acyclic graph* (DAG). A DAG $G = (N, A)$ is a directed graph whose set of vertices is N, whose set of arcs is $A = \{(u, v) \mid u, v \in N\}$ and such that no sequence of arcs from A is a cycle. Given a node $u \in N$, u^+ is the set of parents of u, that is $u^+ = \{v \mid \exists (v, u) \in A\}$.

EXAMPLE 2. *Let us consider two terms $f : x + xy - 1$ and $u : xy$. Terms f and $f[u \leftarrow z]$ can be represented by the following DAGs:*

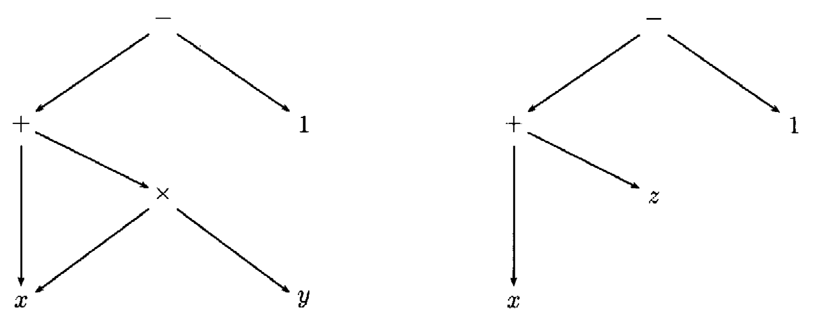


Figure 2.

*Notations:*lower case letters u, v, w denote terms and x, y, z stand for variables. Given a term u and a set of variable domains, the corresponding upper case letter U denotes the interval evaluation of u.

2.1. MULTIDIMENSIONAL INTERVAL NEWTON METHOD

The interval Gauss-Seidel method is an algorithm for bounding the solution set of an interval linear system. In the case of square systems (n variables, n constraints) of nonlinear equations $\boldsymbol{F}(x_1, \ldots, x_n) = \mathbf{0}$, the linear system is classically obtained through a first-order Taylor expansion around the center of the variable domains.

Given an interval vector $\boldsymbol{X}$, $\mathsf{m}(\boldsymbol{X})$ the midpoint of $\boldsymbol{X}$, $\boldsymbol{F}'(\boldsymbol{X})$ an interval extension of the Jacobian matrix of $\boldsymbol{F}$ over $\boldsymbol{X}$, Equation (1) holds:

$$\boldsymbol{F}(\mathsf{m}(\boldsymbol{X})) + \boldsymbol{F}'(\boldsymbol{X})(\boldsymbol{Y} - \mathsf{m}(\boldsymbol{X})) = 0 \tag{1}$$

Let $\boldsymbol{V} = \boldsymbol{Y} - \mathsf{m}(\boldsymbol{X})$ then Equation (1) becomes

$$\boldsymbol{F}'(\boldsymbol{X})\boldsymbol{V} = -\boldsymbol{F}(\mathsf{m}(\boldsymbol{X})). \tag{2}$$

Hansen has shown [9] that if $\boldsymbol{F}'(\boldsymbol{X})$ is not *diagonally dominant* then it is more efficient to perform a preconditioning step —multiplication by a matrix preserving the solution set— transforming Equation (2) into

$$\boldsymbol{P}\boldsymbol{F}'(\boldsymbol{X})\boldsymbol{V} = -\boldsymbol{P}\boldsymbol{F}(\mathsf{m}(\boldsymbol{X})). \tag{3}$$

$\boldsymbol{P}$ is called a *preconditioner*. A standard preconditioner is defined as the inverse of the matrix of midpoints (real numbers) of $\boldsymbol{F}'(\boldsymbol{X})$. The resulting linear system $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{B}$ (Equation (3)) is then solved by applying Gauss-Seidel iterations: the initial value of $\boldsymbol{V}$ is set to $\boldsymbol{W} = \boldsymbol{X} - \mathsf{m}(\boldsymbol{X})$. For i taking values from 1 to n, V_i is computed by evaluating the following formula, assuming that the value of A_{ii} is different from 0 (otherwise V_i takes the value of W_i):

$$V_i = \left(B_i - \sum_{j=1}^{i-1} A_{ij} V_j - \sum_{j=i+1}^{n} A_{ij} W_j \right) / A_{ii}$$

The resulting variable domains are then $\boldsymbol{X} \cap (\boldsymbol{V} + \mathsf{m}(\boldsymbol{X}))$. If an empty domain is derived, then the initial nonlinear system has no solution, as it is guaranteed by the fundamental theorem of interval arithmetic [12]. The whole process is commonly called *multidimensional interval Newton method*.

3. Simultaneous Lazy Evaluation of Partial Derivatives and Constraint Projections

Domain pruning using a multidimensional Newton method combined with the constraint satisfaction algorithm HC4 [1] requires the computation of both partial derivatives [8] and constraint projections. We remark that partial derivatives (computed in backward mode) and constraint projections (by retro-propagation) can be simultaneously evaluated by performing two traversals of the tree-structured representation of functions. Intuitively, a bottom-up traversal evaluates (over $\mathbb{I}$) the ranges of functions for the current variable domains; then, a top-down traversal performs elementary derivation or projection operations. This idea enables a fine-grained cooperation between both algorithms, through the sharing of interval evaluation of terms.

Without loss of generality, we may consider a constraint $c : f \bowtie 0$ where $\bowtie \in \{=, \leqslant, \geqslant\}$. As shown in [8], the computation of partial derivatives ∂_u^f of f is based on the following chain rule:

$$\partial_u^f = \sum_{v \in u^+} \partial_v^f \partial_u^v$$

In the same spirit, we propose to define the following chain rule for computing the partial projections $\pi_u(c)$ of c, where z_v is a fresh variable:

$$\pi_u(c) \subseteq \bigcap_{v \in u^+} \pi_u(v = z_v)$$

The key point is that the conjunction of constraints $c[v \leftarrow z_v] \wedge v = z_v$, where z_v is a fresh variable, is equivalent to c (modulo a cylindrification over $\mathsf{Var}(c) \cup \{z_v\}$). It follows that $v = z_v$ is more general than c. Thus, an outer approximation of the projection of c over u can be computed by projecting the new constraint $v = z_v$ over u. Doing so for each parent v of u, a tight approximation of $\pi_u(c)$ can be obtained as the intersection of all projections $\pi_u(v = z_v)$. Example 3 illustrates this idea:

EXAMPLE 3. *Let $c : x^2 = f(y)$ be a constraint with $x \in X$ and $f(y) \in I$, and let z be a fresh variable. Constraint c can be decomposed into $z = f(y) \wedge x^2 = z$. We have $\pi_x(c) \subseteq \pi_x(x^2 = z)$ since I is necessary included in the domain of z, in other words:*

$$\{a \in X \mid \exists b \in I : a^2 = b\} \subseteq \{a \in X \mid \exists b \in [-\infty, +\infty] : a^2 = b\}$$

We have the following proposition:

PROPOSITION 1. *The chain rule for projections is correct.*

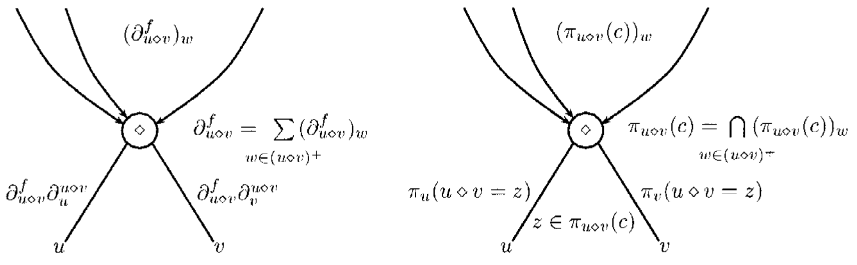
Proof. Let us consider a fresh variable z_v and the conjunction of constraints defined by $c[v \leftarrow z_v] \wedge v = z_v$. This conjunction is considered as a single constraint c' whose relation is the intersection of both relations associated with $c[v \leftarrow z_v]$ and $v = z_v$ and cylindrified over $\mathsf{Var}(c) \cup \{z_v\}$. Since the domain of z_v is exactly the domain of v constrained by c, then c' is equivalent to c (more formally, we must consider the projection of c' over $\mathsf{Var}(c)$), that is $\rho_c = \rho_{c[v \leftarrow z_v]} \cap \rho_{v = z_v}$. Then, we have

$$\pi_u(c[v \leftarrow z_v] \wedge v = z_v) = \pi_u(c).$$

Since $\pi_u(c[v \leftarrow z_v] \wedge v = z_v) \subseteq \pi_u(c[v \leftarrow z_v]) \cap \pi_u(v = z_v)$ (the projection of an intersection is included in the intersection of the projections) and $\pi_u(c[v \leftarrow z_v]) \cap \pi_u(v = z_v) \subseteq \pi_u(v = z_v)$, it follows that $\pi_u(c) \subseteq \pi_u(v = z_v)$. Finally, $\pi_u(c)$ is included in the intersection of the projections $\pi_u(v = z_v)$ associated with all parents v of u, that completes the proof. □

Let us remark that the proof of Proposition 1 only requires the domain of z_v to be a superset of $\pi_v(c)$ (the domain of v constrained by c). This property is weaker than the condition of the domain of a fresh variable to be the set of reals, and will be implemented in the algorithm described thereafter.

Table 1. Backward evaluation of partial derivatives and constraint projections


$(\partial^f_{u\diamond v})_w$
$\partial^f_{u\diamond v} = \sum_{w\in(u\diamond v)^+}(\partial^f_{u\diamond v})_w$
$\partial^f_{u\diamond v}\partial^{u\diamond v}_u$
$\partial^f_{u\diamond v}\partial^{u\diamond v}_v$
u
v
$(\pi_{u\diamond v}(c))_w$
$\pi_{u\diamond v}(c) = \bigcap_{w\in(u\diamond v)^+}(\pi_{u\diamond v}(c))_w$
$\pi_u(u\diamond v = z)$
$\pi_v(u\diamond v = z)$
$z\in\pi_{u\diamond v}(c)$
u
v

PROPOSITION 2. *For any constraint c containing only single occurrences of variables, the chain rule for projections can be replaced with an equality, that is:*

$$\pi_u(c) = \bigcap_{v\in u^+} \pi_u(v = z_v)$$

Proof. In the proof of Proposition 1, the inclusion $\pi_u(c) \subseteq \pi_u(v = z_v)$ is replaced with an equality since u does not appear in $c[v \leftarrow z_v]$ (u is only a sub-term of v which is replaced with z_v). Furthermore, u^+ is reduced to one element for any operation or variable node u in c (only some constants may be shared in the DAG), and then there is only one projection $\pi_u(v = z_v)$ to consider in the intersection. □

The algorithmic process is detailed now. The interval evaluation of every sub-term u of f is a superset of the image of u, which is guaranteed by the fundamental theorem of interval arithmetic [12]. The first stage then consists in a bottom-up traversal of f for computing an outer approximation of the range of every of its sub-term; the range of a term is obtained by applying the corresponding interval operation over the ranges of its sub-terms. The second stage sets either the partial derivative ∂^f_f to 1 or the projection $\pi_f(c)$, which is obtained from the interpretation of relation symbols over $\mathbb{I}$, as follows:

$$\begin{cases} c : f = 0 & \pi_f(c) = [0, 0] \\ c : f \leqslant 0 & \pi_f(c) = [-\infty, 0] \\ c : f \geqslant 0 & \pi_f(c) = [0, +\infty] \end{cases}$$

Finally, the retro-propagation algorithm operates a top-down traversal of f implementing the chain rules. Table 1 illustrates an elementary operation at one node of f containing an operation symbol $\Diamond$. Let us describe the projection operation: the projection $\pi_{u\Diamond v}(c)$ results from the intersection of every projection $(\pi_{u\Diamond v}(c))_w$ already obtained at a parent node w. The contribution of this node to $\pi_u(c)$ (resp. $\pi_v(c)$) is then $\pi_u(u\Diamond v = z)$ (resp. $\pi_v(u\Diamond v = z)$) where z ranges over $\pi_{u\Diamond v}(c)$.

The projections of $u\Diamond v = z$ (resp. derivatives of $u\Diamond v$) are computed by evaluating the inverse operations of $\Diamond$ (resp. elementary derivation rule). The correctness of inverse operations is proved in [5]. Some of those operations are detailed now:

$$\begin{array}{llll}
\partial_u^{u+v} = 1 & & \pi_u(u+v=z) & = Z - V \\
\partial_v^{u+v} = 1 & & \pi_v(u+v=z) & = Z - U \\
\partial_u^{u-v} = 1 & & \pi_u(u-v=z) & = Z + V \\
\partial_v^{u-v} = -1 & & \pi_v(u-v=z) & = U - Z \\
\partial_u^{\log(u)} = 1/U & & \pi_u(\log(u)=z) & = \exp(Z) \\
\partial_u^{\exp(u)} = \exp(U) & & \pi_u(\exp(u)=z) & = \log(Z)
\end{array}$$

The final interesting results are located at variable nodes x, namely reliable approximations for ∂_x^f and for $\pi_x(c)$. Partial derivatives are summed up while projections are intersected. Given $x \in X$, the projection of c over x is computed as:

$$\pi_x(c) = \mathsf{Hull}([\bigcap_{v \in x^+} \pi_x(v=z) \cap X])$$

Since $\pi_x(v=z)$ can be a union of intervals (given non interval convex operations), the hull of the intersection permits to restrict a domain to be an interval. If the new domain of x is empty then this constraint has no solution in the current variable domains. To sum up, this algorithm computes a tighter evaluation for the domain of x after a filtering through constraint c (see Example 4).

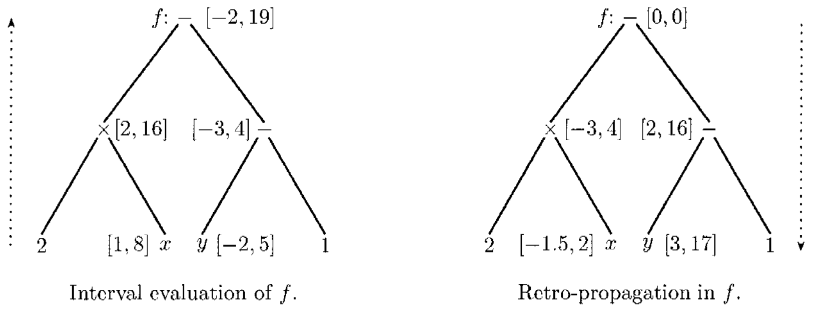
EXAMPLE 4. *Let* $c : 2x = y - 1$ *be a constraint given* $x \in [1, 8]$ *and* $y \in [-2, 5]$. *Table 2 presents the interval evaluation and retro-propagation processes in Term* $f : 2x - (y - 1)$. *The computed approximations for the projections (before intersection with current domains) are* $\pi_x(c) = [-1.5, 2]$ *and* $\pi_y(c) = [3, 17]$. *The new domains are then* $[1, 2] = [1, 8] \cap \pi_x(c)$ *for* x *and* $[3, 5] = [-2, 5] \cap \pi_y(c)$ *for* y.

Retro-propagation at node $\times$ *is detailed now: the projection over* x *of* $2 \times x = z$ *is computed, where* z *is a fresh variable ranging over* $[-3, 4]$ *and* $x \in [1, 8]$. *Constraint* $2 \times x = z$ *is inverted (a symbolic view of this process is the generation of a new constraint* $x = z/2$*) and then* $\pi_x(c) = [1, 8] \cap ([-3, 4]/2) = [1, 2]$.

Let us remark that the projections of a constraint containing only single occurrences of variables are exactly computed by retro-propagation (modulo rounding errors). The reasons are twofold: the projection rule computes an equality in such a case, and the elementary projection operations are exactly implemented since interval evaluation computes the exact ranges of all terms of c (theorem from Moore [12]). As a consequence, only one retro-propagation process permits the computation of all projections, *i.e.* to reach the fixed-point of the algorithm (idempotence property).

For efficiency reasons, interval evaluation of terms can be done during the backward process whenever it is needed (lazy evaluation), as is done by the algorithm computing derivatives in forward mode. More precisely, given the term

Table 2. Computation of constraint projections



$f : u \Diamond v$ considered during retro-propagation on f, the interval evaluation U of u is computed if the three following conditions are verified:

1. Term v contains some variables;
2. Term u contains some operations;
3. U has not been already computed.

Condition 1 means that $\pi_v(c)$ has to be computed. The codes for elementary projection operations suggest that the value U is required (Conditions 2 and 3). Conditions 1 and 2 can be implemented setting one flag at each node of constraint terms. Condition 3 is dynamically evaluated only using one flag during the backward traversal of each term.

The efficiency of lazy evaluation intrinsically depends on the number of variable occurrences in the constraint and their locations in terms. Example 5 illustrates the case where a lazy evaluation is more efficient.

EXAMPLE 5. *Let $c : (x + 1)^3 - 8 = 0$ be a constraint. Retro-propagation successively operates the projection over z_i of the following constraints: $z_1 - 8 = 0$, $z_2^3 = 8$, $z_3 + 1 = 2$. We remark that the interval evaluations of terms of c are not required (see above-mentioned conditions), since all right sub-terms are variable-free. A lazy evaluation then prevents unnecessary computation (here, this is one evaluation of the plus/minus/power operations).*

4. The Branch-and-Prune Algorithm

The chain rule for projections is implemented by Algorithm HC4. It is combined with the Gauss-Seidel method processing a linear system obtained from the nonlinear initial system through a first-order Taylor expansion. The main idea is to first apply HC4 on the original system since the natural interval form – interval evaluation of the original constraint expressions – is efficient for large domains. The Taylor interval form is then generated and processed by Gauss-Seidel.

Table 3 presents the new pruning algorithm. HC4 is enforced over the set of initial constraints $\{c_1, \ldots, c_m\}$, implementing a quasi fixed-point strategy. Algorithm

Table 3. The pruning algorithm

```
Prune($\{c_1, \ldots, c_m\}$; $X = X_1 \times \cdots \times X_n$) : box
begin
  % HC4 algorithm
  Queue := $\{c_1, \ldots, c_m\}$
  while Queue $\neq \varnothing$ and $X \neq \varnothing$ do
    $c$ := first element of Queue
    $X'$ := HC4revise($c, X$)    % computation of the chain rule for projections
    if IsReducedEnough($X', X$) then
      Queue := Queue $\cup \{c_i \mid \exists x_j \in$ Var$(c_i) : X'_j \neq X_j\}$    % propagation
      $X := X'$
    else
      Queue := Queue $\setminus \{c\}$
    endif
  endwhile

  % Gauss-Seidel method
  if $X \neq \varnothing$ then
    $S$ := square system of equations obtained from $\{c_1, \ldots, c_m\}$
    $S'$ := linear system from $S$ through Taylor expansion and preconditioning
      % interval evaluations of terms from HC4 are reused for computing
      % the Jacobian
    $X'$ := Gauss-Seidel($S', X$)
    $X := X \cap (X' + \mathrm{m}(X))$
  endif
  return $X$
end
```

HC4revise applies the chain rule over one constraint c from the queue. If the domains have been contracted enough, all constraints containing a variable whose domain has been modified (condition $X'_j \neq X_j$) are added in that queue. Otherwise, c is removed from the queue, *i.e.* the domains are a quasi fixed-point with respect to c. HC4 stops if the queue or the domains become empty. The Gauss-Seidel method is then enforced over the generated linear system. Let us remark that the Jacobian is efficiently computed since interval evaluations of terms are shared with HC4.

PROPOSITION 3. *For every box X, the new box resulting from the call* Prune $(\{c_1, \ldots, c_m\}, X)$ *is included in X and contains the set of solutions of the system (completeness property), namely $\rho_{c_1} \cap \cdots \cap \rho_{c_m} \cap X$.*

Proof. HC4 and Gauss-Seidel were shown to be contracting and complete [1, 9] (the solution set is preserved). Then so does their combination through intersection of domains. □

As a consequence of Proposition 3, if Prune($\{c_1, \ldots, c_m\}$,$\boldsymbol{X}$) is empty, so is the solution set.

Table 4. The stronger pruning algorithm

```
StrongPrune($C = \{c_1, \ldots, c_m\}$; $\boldsymbol{X} = X_1 \times \cdots \times X_n$; $w$) : box
% $w \in \mathbb{R}^+$ is the precision of the algorithm
begin
  modified := true
  while modified and $\boldsymbol{X} \neq \varnothing$ do
      $\boldsymbol{X}$ := Prune($C, \boldsymbol{X}$)   % contraction of $\boldsymbol{X}$ by Algorithm Prune
      modified := false
      for $i$ := 1 to $n$ do   % local search for the contraction of $X_i$
          let $X_i = [a, b]$
          $I$ := $X_i$          % the new domain is computed in $I$
          $X_i$ := $[a, a + w] \cap I$
          if Prune($C, \boldsymbol{X}$) $= \varnothing$ then
             $I$ := $I \cap [a + w, b]$   % $[a, a + w[$ is removed from $X_i$
             modified := true
          endif
          $X_i$ := $[b - w, b] \cap I$
          if Prune($C, \boldsymbol{X}$) $= \varnothing$ then
             $I$ := $I \cap [a, b - w]$   % $]b - w, b]$ is removed from $X_i$
             modified := true
          endif
          $X_i$ := $I$
      endfor
  endwhile
  return $\boldsymbol{X}$
end
```

As in [15], Algorithm Prune is combined with a local search over the domain of one variable (see [11] for a presentation of strong consistencies). That defines a stronger pruning algorithm called StrongPrune and described in Table 4. The idea is to prove the inconsistency of a small interval at one bound of one variable domain ($[a, a + w]$ or $[b - w, b]$) deriving the empty domains by Prune. In that case, this subdomain can be removed due to the completeness property of Prune (Proposition 3).

PROPOSITION 4. *For every box $\boldsymbol{X}$, the box computed by the call* StrongPrune ($\{c_1, \ldots, c_m\}$,$\boldsymbol{X}$) *is included in $\boldsymbol{X}$ and contains the set of solutions of the system, namely $\rho_{c_1} \cap \cdots \cap \rho_{c_m} \cap \boldsymbol{X}$.*

The completeness of Algorithm StrongPrune comes from the completeness of Prune. The contraction property trivially holds.

Algorithm StrongPrune is embedded in a branch-and-prune algorithm splitting the domains if the desired precision has not been reached and reinvoking

StrongPrune. The result is a set of interval vectors —variable domains— such that the union of all corresponding Cartesian products contains the solutions of the nonlinear system.

5. The Circuit Design Problem

The circuit design problem from Ebers and Moll is described by the following nonlinear system:

$$\begin{cases} (1 - x_1x_2)x_3[\exp(x_5(g_{1k} - g_{3k}x_7 10^{-3} - g_{5k}x_8 10^{-3})) - 1] \\ \qquad -g_{5k} + g_{4k}x_2 = 0 \quad (1 \leqslant k \leqslant 4) \\ (1 - x_1x_2)x_4[\exp(x_6(g_{1k} - g_{2k} - g_{3k}x_7 10^{-3} + g_{4k}x_9 10^{-3})) - 1] \\ \qquad -g_{5k}x_1 + g_{4k} = 0 \quad (1 \leqslant k \leqslant 4) \\ x_1x_3 - x_2x_4 = 0 \\ x_k \in [0, 10] \quad (1 \leqslant k \leqslant 9) \end{cases}$$

The constants g_{ik} are given by the following matrix:

$$\begin{pmatrix} 0.4850 & 0.7520 & 0.8690 & 0.9820 \\ 0.3690 & 1.2540 & 0.7030 & 1.4550 \\ 5.2095 & 10.0677 & 22.9274 & 20.2153 \\ 23.3037 & 101.7790 & 111.4610 & 191.2670 \\ 28.5132 & 111.8467 & 134.3884 & 211.4823 \end{pmatrix}$$

The problem is to prove that the box has exactly one solution of the equations, which shall be computed as precise as possible.

The branch-and-prune algorithm has been applied on this problem. A domain is bisected if its width is greater than 10^{-12}. Parameter w of Algorithm StrongPrune is set to 0.005. Function IsReducedEnough of Algorithm Prune succeeds if the improvement of the width of domains is more than 10%. The result is a single box, proved to contain a unique solution (see [15] for more details):

$$\begin{array}{lcllll} x_1 & = & 0.899999952617 & + & [-3.331\text{e}{-16}, & +3.331\text{e}{-16}] \\ x_2 & = & 0.449987471982 & + & [-3.308\text{e}{-14}, & +3.314\text{e}{-14}] \\ x_3 & = & 1.00000648247 & + & [-8.304\text{e}{-14}, & +8.304\text{e}{-14}] \\ x_4 & = & 2.00006854162 & + & [-9.059\text{e}{-14}, & +9.104\text{e}{-14}] \\ x_5 & = & 7.99997144051 & + & [-1.572\text{e}{-13}, & +1.572\text{e}{-13}] \\ x_6 & = & 7.99969268422 & + & [-2.700\text{e}{-13}, & +2.709\text{e}{-13}] \\ x_7 & = & 5.00003127593 & + & [-1.252\text{e}{-13}, & +1.261\text{e}{-13}] \\ x_8 & = & 0.999987723457 & + & [-1.421\text{e}{-14}, & +1.432\text{e}{-14}] \\ x_9 & = & 2.00005248349 & + & [-6.484\text{e}{-14}, & +6.528\text{e}{-14}] \end{array}$$

The whole process is 444s long on a Sun Sparc UltraII/166 MHz and requires 81 bisections; the solution is computed after 220s. Puget and Van Hentenryck' method performs 118 bisections to derive the solution with a precision of 10^{-8} and takes 2360s, what is about five times slower than our algorithm. The improvement in time results from the implementation of the chain rule for projections instead of box consistency, the cooperation with the multidimensional Newton method, and the stopping criterion of domain pruning (no fixed-point is computed). The improvement in bisections (that corresponds to a best precision of pruning) probably comes from the use of different values of parameter w. Let us finally remark that the chain rule for projections is replaced with an equality since all variables occur once in each constraint, and then is optimal for this problem. In other words, HC4 is here faster than and as precise as the classical algorithm for box consistency.

6. Conclusion

This paper proposes to modify Puget and Van Hentenryck' constraint satisfaction algorithm [15] by replacing box consistency with hull consistency computed with algorithm HC4 [1]. HC4 is shown to implement a chain rule for constraint projections, and to be idempotent for constraints composed of single occurrences of variables. A precise solution for the circuit design problem from Ebers and Moll [6] is then efficiently computed.

An interesting direction for future research is the transformation of the original problem into a polynomial one, introducing new variables for terms containing the exponential operation, or generating a Taylor expansion of the exponential function [4]. The polynomial problem could be simplified, using Gröbner bases computations, and then solved by interval techniques [2].

Notes

[1]Each coefficient (ij) of the matrix $\boldsymbol{F'}(\boldsymbol{X})$ is a superset of the range of the partial derivative $\partial_{x_j}^{f_i}$ over $\boldsymbol{X}$.
[2]Every constraint $g \bowtie h$ can be transformed beforehand into $g - h \bowtie 0$.
[3]That corresponds in [15] to the computation of *bound consistency*.

Acknowledgements

The research exposed here was supported in part by the project 98/06 supported by the French/Russian A.M. Liapunov Institute.

References

1. Benhamou, F., Goualard, F., Granvilliers, L. and Puget, J.-F. (1999), Revising Hull and Box Consistency. In *Proc. International Conference on Logic Programming*, Las Cruces, USA. The MIT Press.
2. Benhamou, F. and Granvilliers, L. (1997), Automatic Generation of Numerical Redundancies for Non-Linear Constraint Solving. *Reliab. Comput.* 3(3): 335–344.
3. Benhamou, F. and Older, W. J. (1997), Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *J. Logic Programming* 32(1): 1–24.

4. Berz, M. and Hoffstätter, G. (1998), Computation and Application of Taylor Polynomials with Interval Remainder Bounds. *Reliab. Comput.* 4: 83–97.
5. Cleary, J. G. (1987), Logical Arithmetic. *Future Computing Systems* 2(2): 125–149.
6. Ebers, J. J. and Moll, J. L. (1954), Large-Scale Behaviour of Junction Transistors. In *IEE. Proc.*, volume 42, pages 1761–1772.
7. Faltings, B. (1994), Arc Consistency for Continuous Variables. *Artificial Intelligence* 65(2): 363–376.
8. Griewank, A. (1989), On automatic differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108.
9. Hansen, E. R. (1992), *Global Optimization using Interval Analysis*. Marcel Dekker.
10. Hyvönen, E. (1992), Constraint Reasoning based on Interval Arithmetic. The Tolerance Propagation Approach. *Artificial Intelligence* 58: 71–112.
11. Lhomme, O. (1993), Consistency Techniques for Numeric CSPs. In *Proc. International Joint Conference of Artificial Intelligence*, Chambéry, France. Morgan Kaufmann.
12. Moore, R. E. (1966), *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ.
13. Neumaier, A. (1990), *Interval Methods for Systems of Equations*. Cambridge University Press.
14. Older, W. and Vellino, A. (1993), Constraint Arithmetic on Real Intervals. In F. Benhamou and A. Colmerauer (eds.), *Constraint Logic Programming: Selected Research*. MIT Press.
15. Puget, J.-F. and Van Hentenryck, P. (1998), A Constraint Satisfaction Approach to a Circuit Design Problem. *J. Global Optim.* 13(1): 75–93.
16. Ratschek, H. and Rokne, J. (1993), Experiments using Interval Analysis for Solving a Circuit Design Problem. *J. Global Optim.* 3: 501–518.
17. 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.
18. Van Hentenryck, P., Michel, L. and Deville, Y. (1997), *Numerica: a Modeling Language for Global Optimization*. MIT Press.