Journal of Global Optimization **13:** 75–93, 1998.

A Constraint Satisfaction Approach to a Circuit Design Problem

JEAN-FRANÇOIS PUGET[1] and PASCAL VAN HENTENRYCK[2]
[1]*Ilog SA, 9 rue de Verdun, F-94253 Gentilly, France; E-mail: puget@ilog.fr*
[2]*Brown University, Box 1910, Providence, RI 02912, USA; E-mail: pvh@cs.brown.edu*

(Received 23 December 1996; accepted 28 December 1997)

Abstract. A classical circuit-design problem from Ebers and Moll (1954) features a system of nine nonlinear equations in nine variables that is very challenging both for local and global methods. This system was solved globally using an interval method by Ratschek and Rokne (1993) in the box $[0, 10]^9$. Their algorithm had enormous costs (i.e., over 14 months using a network of 30 Sun Sparc-1 workstations) but they state that "at this time, we know no other method which has been applied to this circuit design problem and which has led to the same guaranteed result of locating exactly one solution in this huge domain, completed with a reliable error estimate". The present paper gives a novel branch-and-prune algorithm that obtains a unique safe box for the above system within reasonable computation times. The algorithm combines traditional interval techniques with an adaptation of discrete constraint-satisfaction techniques to continuous problems. Of particular interest is the simplicity of the approach.

Key words: Global zero search, Electrical circuit, Transistor modelling, Interval methods, Branch and prune, Constraint satisfaction

1. Introduction

The transistor modelling problem of Ebers and Moll [6] is the system of nonlinear equations

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

where the constants g_{ik} are given by

0.485	0.752	0.869	0.982
0.369	1.254	0.703	1.455
5.2095	10.0677	22.9274	20.2153
23.3037	101.779	111.461	191.267
28.5132	111.8467	134.3884	211.4823

The problem is very challenging both for local and global methods because small variations in the inputs produce large differences in the functions. Ratschek and Rokne [23] summarize various attempts to find a solution to this problem using local methods; these descriptions are not repeated here. It suffices to say that successful attempts require very elaborate procedures, sometimes combining several globally convergent algorithms.

In the same paper [23], Ratschek and Rokne propose an interval method which solves the problem globally in the box $[0, 10]^9$. In particular, they show that there exists a unique solution in this box and they provide a guaranteed error estimate by enclosing the solution in a box whose intervals are of width smaller than 3.2×10^{-4}. The computation times to obtain this result are, however, extremely large. No computation times were given in [23], since this was not the primary aim of the paper. However, a preliminary draft of the paper indicated that the total process took over 14 months using a network of 30 Sun Sparc-1 workstations. Ratschek and Rokne also recommended that such methods for solving unstable systems should be further investigated in order to reduce the computations.

Here we present a novel interval algorithm that isolates all safe boxes (i.e., all boxes which contain at least one solution) to nonlinear equation systems and show that the algorithm isolates a single safe box for the circuit-design problem within reasonable computation times.* The main novelty of the procedure is in the way in which constraints are used to prune the search space. The pruning techniques, some of which were presented in [24] and some of which are novel, are based on constraint- satisfaction techniques from artificial intelligence and are particularly effective when far from a solution. These techniques are thus orthogonal to traditional interval techniques, which are most effective close to a solution (when the boxes are small).

The paper is organized as follows. Section 2 gives the necessary background in interval analysis. Section 3 discusses the problem description and the simplyfying assumptions. Section 4 presents a generic branch-and-prune algorithm for isolating all solutions to a nonlinear system of equations, Section 5 presents the various pruning techniques, and Section 6 reconsiders the simplifying assumptions. Section 7 reports the experimental results and Section 8 concludes the paper.

2. Preliminaries

Here we review some basic concepts of interval analysis to needed for this paper. More information on interval arithmetic can be found in many places [e.g., 1, 7, 8, 17, 18, 20, 22]. Our definitions are slightly non-standard.

* Proving that the safe box contains a unique solution requires some experimental trial-and-error work; see Section 6 of Ratschek and Rokne [23] for a discussion of this issue.

2.1. INTERVAL ARITHMETIC

We consider $\Re^\infty = \Re \cup \{-\infty, \infty\}$, the set of real numbers extended with the two infinity symbols, and the extension of the relation $<$ to this set. We also consider a finite subset $\mathcal{F}$ of $\Re^\infty$ containing $-\infty, \infty, 0$. In practice, $\mathcal{F}$ corresponds to the floating-point numbers used in the implementation.

DEFINITION 1 (Interval). *An interval $[l, u]$ with $l, u \in \mathcal{F}$ is the set of real numbers*

$$\{r \in \Re | l \leqslant r \leqslant u\}.$$

The set of intervals is denoted by $\mathbb{I}$ and is ordered by set inclusion.⋆

DEFINITION 2 (Enclosure). *Let S be a subset of $\Re$. The enclosure of S, denoted by $\overline{S}$, or $\Box S$, is the smallest interval I such that $S \subseteq I$. We often write $\overline{r}$ instead of $\overline{\{r\}}$ for $r \in \Re$.*

We denote real numbers by the letters r, v, a, b, c, d, $\mathcal{F}$-numbers by the letters l, m, u, intervals by the letter I, real functions by the letters f, g and interval functions (e.g., functions of signature $\mathbb{I} \to \mathbb{I}$) by the letters F, G, all possibly subscripted. We use l^+ (resp. l^-) to denote the smallest (rep. largest) $\mathcal{F}$- number strictly greater (resp. smaller) than the $\mathcal{F}$-number l. To capture outward rounding, we use $\lceil r \rceil$ (resp. $\lfloor r \rfloor$) to return the smallest (resp. largest) $\mathcal{F}$-number greater (resp. smaller) or equal to the real number r. We also use $\vec{I}$ to denote a box $\langle I_1, \ldots, I_n \rangle$ and $\vec{r}$ to denote a tuple $\langle r_1, \ldots, r_n \rangle$. $\mathcal{Q}$ is the set of rational numbers and $\mathcal{N}$ is the set of natural numbers. We also use the following notations:

$$\begin{aligned} \text{left}([l, u]) &= l \\ \text{right}([l, u]) &= u \\ \text{center}([a, b]) &= \lfloor (a + b)/2 \rfloor \end{aligned}$$

DEFINITION 3 (Canonical interval). *A canonical interval is an interval of the form $[l, l]$ or $[l, l^+]$, where l is a floating-point number.*

The fundamental concept of interval arithmetic is the notion of interval extension.

DEFINITION 4 (Set extensions). *Consider a set $S \subseteq \Re^n$ and a function $f : \Re^n \to \Re$. The set extension of f is defined as*

$$f(S) = \{f(\vec{r}) | \vec{r} \in S\}.$$

DEFINITION 5 (Interval extensions). *An interval function $F : \mathbb{I}^n \to \mathbb{I}$ is an interval extension of $f : \Re^n \to \Re$ in $\vec{I}_0$ if*

$$\forall \vec{I} \subseteq \vec{I}_0 : f(\vec{I}) \subseteq F(\vec{I}).$$

⋆ These intervals are usually called floating-point intervals in the literature.

EXAMPLE 1 *The interval function* $\oplus$ *defined as*

$$[a_1, b_1] \oplus [a_2, b_2] = [\lfloor a_1 + a_2 \rfloor, \lceil b_1 + b_2 \rceil]$$

is an interval extension of addition of real numbers.

In this paper, we restrict attention to monotonic interval extensions because of their fundamental properties and because traditional interval extensions of primitive operations satisfy these requirements naturally.

DEFINITION 6 (Monotonic interval extensions). *An interval function* $F : \mathbb{I}^n \rightarrow \mathbb{I}$ *is monotonic in* $\vec{I}_0$ *if*

$$\forall \vec{I}_1, \vec{I}_2 \subseteq \vec{I}_0 : \vec{I}_1 \subseteq \vec{I}_2 \Rightarrow F(\vec{I}_1) \subseteq F(\vec{I}_2).$$

It is important to stress that a real function can be extended in many ways. For instance, the interval function $\oplus$ is the most precise interval extension of addition (i.e., it returns the smallest possible interval containing all real results), while a function always returning $[-\infty, \infty]$ would be the least accurate. In the following, we assume fixed monotonic interval extensions for the primitive operators (for instance, the interval extension of $+$ is defined by $\oplus$). In addition, we overload the real symbols and use them for their interval extensions.

2.2. CONSTRAINT REPRESENTATIONS

It is well known that different computer representations of a real function produce different results when evaluated with floating-point numbers on a computer. As a consequence, the way in which constraints are written may have an impact on the behavior on the algorithm. For this reason, a constraint or a function in this paper is considered to be an expression written in some language. In addition, we abuse notation by denoting functions (resp. constraints) and their representations by the same symbol. In the remaining sections, we assume that real variables in constraints are taken from a finite (but arbitrarily large) set $\{x_1, \ldots, x_n\}$. Similar conventions apply to interval functions and constraints. Interval variables are taken from a finite (but arbitrarily large) set $\{X_1, \ldots, X_n\}$.

3. Problem description

The problem considered in this paper is finding all solutions to a system of equations

$$S = \begin{cases} 0 = f_1(x_1, \ldots, x_n) \\ \ldots \\ 0 = f_n(x_1, \ldots, x_n) \end{cases}$$

in a box $\vec{I}^0 = \langle I_1^0, \ldots, I_n^0 \rangle$. Of course, on a computer, it is generally impossible to find these solutions exactly and interval methods aim at returning small boxes containing all solutions. Preferably, each such box is *safe*, meaning that it contains a solution. For the purposes of this paper, interval methods can thus be viewed as solving the following problem: assuming that F_i is a monotonic interval extension of $f_i (1 \leqslant i \leqslant n)$, find all canonical* boxes $\langle I_1, \ldots, I_n \rangle$ in $\vec{I}^0$ satisfying

$$\mathcal{S} = \begin{cases} 0 \in F_1(I_1, \ldots, I_n) \\ \ldots \\ 0 \in F_n(I_1, \ldots, I_n) \end{cases}$$

This is of course a simplification, since interval methods generally use various interval extensions. However, restricting attention to this problem has the benefits of crystallizing the intuition underlying our novel pruning methods. Section 6 reconsiders this simplification.

NOTATION: Let $\mathcal{S}$ be a system of constraints of the form

$$\mathcal{S} = \begin{cases} 0 \in F_1(X_1, \ldots, X_n) \\ \ldots \\ 0 \in F_n(X_1, \ldots, X_n) \end{cases}$$

and let $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$. We denote by $\mathcal{S}(\vec{I})$ and $\mathcal{S}(I_1, \ldots, I_n)$ the fact that $\vec{I}$ satisfies the system of interval constraints $\mathcal{S}$ or, in symbols,

$$0 \in F_1(I_1, \ldots, I_n) \wedge \cdots \wedge 0 \in F_n(I_n, \ldots, I_n).$$

Note also that we use S to denote systems of constraints and $\mathcal{S}$ to denote systems of interval constraints.

4. The generic branch-and-prune algorithm

The above problem description highlights the finite nature of the problem, since there are only finitely many floating-point intervals. Most interval methods are thus best viewed as global search algorithms iterating two main steps: pruning and branching. The basis schema underlying these algorithms, the branch-and-prune schema, is depicted in Figure 1. The function `Search` receives a system of interval constraints $\mathcal{S}$ and an initial box $\vec{I}^0$: it returns the set of canonical boxes $\vec{I}$ in $\vec{I}^0$ satisfying $\mathcal{S}(\vec{I})$. The function `Search` first applies a pruning step that reduces the initial box. This pruning step is the main topic of this paper. If the resulting box $\vec{I}$ is empty, there is no solution to the problem. If the box $\vec{I}$ is canonical, it is returned

* In practice, one may be interested in boxes of a certain width or one may want to stop as soon as a safe box is obtained. It is easy to generalize our results to these requirements.

```
function Search(𝒮, Ī₀)
begin
  Ī := Prune(𝒮, Ī₀) :
  if Empty(Ī) then
    return ∅
  else if Canonical(Ī) then
    return {Ī}
  else
    ⟨Ī₁, Ī₂⟩ := Split(Ī);
    return Search(𝒮, Ī₁) ∪ Search(𝒮, Ī₂);
end
```

Figure 1. The branch-and-prune algorithm.

as a result. Otherwise, the box is split along one dimension into two subboxes, $\vec{I}_1$ and $\vec{I}_2$, which are then explored recursively using the same algorithm.

A specific interval algorithm can be obtained by specifying the splitting strategy and pruning techniques. Our algorithms use a splitting strategy that consists of splitting the largest interval in two parts. The main novelty of our algorithms lies in the pruning techniques and we will define three pruning operators, $\texttt{Prune}_0$, $\texttt{Prune}_1$, $\texttt{Prune}_2$, that produce three distinct algorithms.

5. The pruning techniques

The two pruning techniques used in our algorithm are box(1)- and box(2)-consistency. To ease understanding, these techniques are contrasted with a traditional interval technique, which we call box(0)-consistency for reasons that will become clear below.

5.1. BOX(0)-CONSISTENCY

It is traditional in branch-and-prune algorithms to use a relaxation of the problem at hand. If there is no solution to the easier problem, it follows that there are no solutions to the original problem. Box(0)-consistency is a weak, but very simple, relaxation used in most interval systems. Given the problem of finding canonical boxes in a box $\vec{I}$ satisfying a system of interval constraints $\mathcal{S}$, box(0)-consistency consists of testing $\mathcal{S}(\vec{I})$. If $\mathcal{S}(\vec{I})$ does not hold, there are obviously no solutions to the original problem, because of the definition of interval extensions. The pruning

operator associated with box(0)-consistency can thus be defined as follows:

$$\texttt{Prune}_0(\mathcal{S}, \vec{I}) = \begin{cases} \emptyset & \text{if } \neg\mathcal{S}(\vec{I}) \\ \vec{I} & \text{otherwise.} \end{cases}$$

Box(0)-consistency can in fact be viewed as a form of projection. The original problem could be stated as an existence question

$$\exists X_1 \subseteq I_1, \ldots, \exists X_n \subseteq I_n : \mathcal{S}(X_1, \ldots, X_n)$$

and box(0)-consistency approximates it by replacing each interval variable by its interval to obtain the test

$$\mathcal{S}(I_1, \ldots, I_n)$$

which reduces to testing each constraint in $\mathcal{S}$ independently.

5.2. BOX(1)-CONSISTENCY

This section presents the first pruning operator used in our algorithm. It starts with an informal discussion, then specifies the pruning operator, and presents a simple implementation.*

5.2.1. *Informal presentation*

The first fundamental idea underlying box(1)-consistency [2] is to project all variables but one or, more precisely, to replace all variables but one by their intervals. This produces a stronger pruning than box(0)-consistency but, of course, at a higher cost. The original existence problem

$$\exists X_1 \subseteq I_1, \ldots, \exists X_n \subseteq I_n : \mathcal{S}(X_1, \ldots, X_n)$$

is thus approximated by

$$\begin{aligned} &\exists X_1 \subseteq I_1 : \mathcal{S}(X_1, I_2, \ldots, I_n) \wedge \\ &\exists X_2 \subseteq I_2 : \mathcal{S}(I_1, X_2, \ldots, I_n) \wedge \\ &\ldots \\ &\exists X_n \subseteq I_n : \mathcal{S}(I_1, \ldots, I_{n-1}, X_n) \end{aligned}$$

This relaxation can be tested relatively easily. Notice first that the conditions are independent. In addition, a condition of the form

$$\exists X_1 \subseteq I_1 : \mathcal{S}(X_1, I_2, \ldots, I_n)$$

* Separating the specification from the implementation has the advantage of distinguishing what is being computed from how to compute it. There are many ways to implement the concepts described in this section, and our goal here is to focus on the concepts, not on the technical details (which can be found in Van Hentenryck et al. [24]).

can be tested by considering all the canonical intervals I in I_1 and testing

$$\mathcal{S}(I, I_2, \ldots, I_n)$$

Our implementation tries to be more effective by using adaptations of the interval Newton method.

The second fundamental idea underlying box(1)-consistency is to reduce the intervals associated with the variables. Consider the relaxation

$$\exists X_1 \subseteq I_1 : \mathcal{S}(X_1, I_2, \ldots, I_n)$$

and let I_l be the leftmost interval in I_1 satisfying

$$\mathcal{S}(I_l, I_2, \ldots, I_n)$$

and I_r the rightmost interval in I_1 satisfying

$$\mathcal{S}(I_r, I_2, \ldots, I_n).$$

It should be clear that X_1 must range in the interval I'

$$I' = [\text{left}(I_l),\ \text{right}(I_r)]$$

since any interval on the left or on the right of I' violates one of the conditions of the relaxation. The interval associated with X_1 can thus be reduced to I'.

This reduction is applied for each of the variables until no more reduction takes place. The resulting box is said to be *box(1)-consistent*. In the course of this process, it is possible to detect that no solution to the original problem exists, since the intervals associated with the variables become smaller. Note also that a variable can be considered several times in this reduction process.

5.2.2. *The pruning operator*

We now formalize the informal discussion above and present the pruning operator associated with box(1)-consistency. Recall that all definitions assume that $\mathcal{S}$ is defined over the set of variables $X_1, \ldots, X_n$. The main concept is box(1)-consistency, which expresses that a system cannot be narrowed further by the reduction process described informally in the previous subsection.

DEFINITION 7 (Box(1)-consistency). *Let $\mathcal{S}$ be a system of interval constraints, let $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$, and let $l_i = left(I_i)$ and $u_i = right(I_I)$ $(1 \leqslant i \leqslant n)$. $\mathcal{S}$ is box(1)-consistent in $\vec{I}$ wrt i if*

$$\mathcal{S}(I_1, \ldots, I_{i-1}, [l_i, l_i^+], I_{i+1}, , \ldots, I_n)$$
$$\wedge\ \mathcal{S}(I_1, \ldots, I_{i-1}, [u_i^-, u_i], I_{i+1}, \ldots, I_n) \text{ when } l_i \neq u_i$$

and

$$\mathcal{S}(I_1, \ldots, I_{i-1}, [l_i, l_i], I_{i+1}, \ldots, I_n) \text{ when } l_i = u_i.$$

$\mathcal{S}$ is box-consistent in $\vec{I}$ if it is box(1)-consistent in $\vec{I}$ wrt all i in $1 \ldots n$.

```
function Prune_1(S, I⃗)
begin
  repeat
    I⃗_p := I⃗;
    I⃗ = ∩{Narrow_1(S, I⃗, i) | 1 ⩽ i ⩽ n};
  until I⃗ = I⃗_p;
  return I⃗;
end

function Narrow_1(S, ⟨I_1, ..., I_n⟩, i)
begin
  C := {I_c ⊆ I_1 | I_c is canonical and S(I_1, ..., I_{i-1}, I_c, I_{i+1}, ..., I_n)};
  if C = ∅ then
    return ∅;
  else
    return [min_{i∈C} left(I), max_{I∈C} right(I)];
end
```

Figure 2. Implementing box(1)-consistency.

The pruning operator associated with box(1)-consistency simply returns the largest box in the initial interval that is box(1)-consistent (or, more informally, the largest box in the initial interval that cannot be narrowed further). This box always exists because of the monotonicity of interval extensions and is unique. Of course, it can be empty.

DEFINITION 8 (Box(1)-consistency pruning). *Let $\mathcal{S}$ be a system of interval constraints and let $\vec{I}$ be a box. The pruning operator associated with box(1)-consistency can be defined as*

$$\mathtt{Prune}_1(\mathcal{S}, \vec{I}) = \vec{I}'$$

where $\vec{I}'$ is the largest box in $\vec{I}$ such that $\mathcal{S}$ is box(1)- consistent in $\vec{I}'$.

5.2.3. *A simple implementation*

The pruning operator can be computed in many ways. Figure 2 presents a simple iterative algorithm for this purpose; see Van Hentenryck [24] for a more efficient implementation. The algorithm is a simple fixpoint algorithm that terminates when no further pruning can be obtained (i.e., $\vec{I} = \vec{I}_p$). The body of the loop applies a narrowing operation on each of the variables and produces a new box that is the intersection of all these narrowings. The narrowing function $\mathtt{Narrow}_1(\mathcal{S}, \vec{I}, i)$ returns the largest box $\vec{I}'$ in $\vec{I}$ such that $\mathcal{S}$ is box(1)-consistent in $\vec{I}'$ wrt i.

5.3. BOX(2)-CONSISTENCY

Box consistency has been shown to be effective for solving a variety of nonlinear applications [24]. For some applications, however, and for the transistor modelling problem in particular, better performance can be obtained by using a stronger local consistency condition that we call *box(2)-consistency*. Box(2)-consistency is in fact an approximation of path consistency [16] and is related to some consistency notions presented in L'Homme [14].

5.3.1. *Informal presentation*

Box(2)-consistency generalizes box(1)-consistency by projecting all but two variables. The original existence problem

$$\exists X_1 \subseteq I_1, \ldots, \exists X_n \subseteq I_n : \mathcal{S}(X_1, \ldots, X_n)$$

is thus approximated by

$$\begin{aligned}
&\exists X_1 \subseteq I_1, X_2 \subseteq I_2 : \mathcal{S}(X_1, X_2, I_3, \ldots, I_n) \wedge \\
&\exists X_1 \subseteq I_1, X_3 \subseteq I_3 : \mathcal{S}(X_1, I_2, X_3, I_4, \ldots, I_n) \wedge \\
&\ldots \\
&\exists X_2 \subseteq I_2, X_3 \subseteq I_3 : \mathcal{S}(I_1, X_2, X_3, I_4, \ldots, I_n) \wedge \\
&\ldots \\
&\exists X_{n-1} \subseteq I_{n-1}, X_n \subseteq I_n : \mathcal{S}(I_1, \ldots, I_{n-2}, X_{n-1}, X_n)
\end{aligned}$$

Once again, it is possible to test this relaxation easily, at least from a conceptual standpoint. The conditions are independent and a condition of the form

$$\exists X_1 \subseteq I_1, X_2 \subseteq I_2 : \mathcal{S}(X_1, X_2, I_3, \ldots, I_n)$$

can be tested by considering all pairs of canonical intervals in I_i and I_2 for X_1 and X_2.

As was the case for box(1)-consistency, box(2)-consistency makes use of this relaxation to prune the intervals associated with each variable. Consider a condition

$$\exists X_1 \subseteq I_1, X_2 \subseteq I_2 : \mathcal{S}(X_1, X_2, I_3, \ldots, I_n)$$

and a canonical interval $I'_1 \subseteq I_1$. If there is no box $\vec{I}' \subseteq \langle I'_1, I_2, \ldots, I_n \rangle$ such that $\mathcal{S}$ is box(1)-consistent in $\vec{I}'$, then obviously $x \notin I'_1$. It is thus possible to narrow the bounds of I_1 by using this pruning rule, and this process can be iterated for all variables until no further pruning is available.

5.3.2. *The pruning operator*

The notion of box(2)-consistency is defined in terms of box(1)-consistency or, more precisely, in terms of whether a system of interval constraints can be made box(1)-consistent in a box.

DEFINITION 9 (Box(1)-satisfiability). *A system of interval constraints $\mathcal{S}$ is box(1)-satisfiable in $\vec{I}_0$, denoted by $BoxSat_1(\mathcal{S}, \vec{I}_0)$, if there exists a box $\vec{I} \subseteq \vec{I}_0$ such that $\mathcal{S}$ is box(1)-consistent in $\vec{I}$.*

Informally speaking, box(2)-consistency says that the bounds of each variable are box-satisfiable, implying that they cannot be reduced further using the pruning rule described above.

DEFINITION 10 (Box(2)-consistency). *Let $\mathcal{S}$ be a system of interval constraints and $\vec{I}$ be a box $\langle I_1, \ldots, I_n\rangle$ with $I_j = [l_j, u_j]$. $\mathcal{S}$ is box(2)-consistent in $\vec{I}$ wrt i if*

$$BoxSat_1(\mathcal{S}, \langle I_1, \ldots, I_{i-1}, [l_i, l_i^+], I_{i+1}, \ldots, I_n\rangle) \wedge$$
$$BoxSat_1(\mathcal{S}, \langle I_1, \ldots, I_{i-1}, [u_i^-, u_i], I_{i+1}, \ldots, I_n\rangle)$$

when $l_i \neq u_i$ and if

$$BoxSat_1(\mathcal{S}, \vec{I}) \text{ otherwise.}$$

The system $\mathcal{S}$ is box(2)-consistent in $\vec{I}$ if it is box(2)-consistent in $\vec{I}$ wrt all $i (1 \leqslant i \leqslant n)$.

It can be shown that box(2)-consistency implies box(1)-consistency.

PROPOSITION 1. *Let $\mathcal{S}$ be a system of monotonic interval constraints. If $\mathcal{S}$ is box(2)-consistent in $\vec{I}$, then $\mathcal{S}$ is box(1)-consistent in $\vec{I}$.*

Proof. Assume for simplicity that $\vec{I} = \langle I_1, \ldots, I_n\rangle$ with $l_i \neq u_i$. The proof is similar otherwise. Since $\mathcal{S}$ is box(2)-consistent in $\vec{I}$, $\mathcal{S}$ is box(2)-consistent in $\vec{I}$ wrt all $i (1 \leqslant i \leqslant n)$. Thus,

$$BoxSat_1(\mathcal{S}, \langle I_1, \ldots, I_{i-1}, [l_i, l_i^+], I_{i+1}, \ldots, I_n\rangle)$$

or, more explicitly,

$$\exists \vec{I}' \subseteq \langle I_1, \ldots, I_{i-1}, [l_i, l_i^+], I_{i+1}, \ldots, I_n\rangle : \mathcal{S} \text{ is box(1)-consistent in } \vec{I}'.$$

It follows that

$$0 \in F_i(\vec{I}') \quad (1 \leqslant i \leqslant n)$$

and, by monotonicity of F_i, that

$$0 \in F_i(I_1, \ldots, I_{i-1}, [l_i, l_i^+], I_{i+1}, \ldots, I_n).$$

The proof that

$$0 \in F_i(I_1, \ldots, I_{i-1}, [u_i^-, u_i], I_{i+1}, \ldots, I_n)$$

is similar. The result follows from the definition of box(1)-consistency. □

The pruning operator associated with box(2)-consistency simply returns the largest box in the initial interval which is box(2)-consistent.

```
function Prune2(𝒮, Ī)
begin
  repeat
    Ī_p := Ī;
    Ī = ∩{Narrow2(𝒮, Ī, i) | 1 ⩽ i ⩽ n};
  until Ī = Ī_p;
  return Ī;
end

function Narrow2(𝒮, ⟨I_1, ..., I_n⟩, i)
begin
  C := {I_c ⊆ I_i | I_c is canonical and ¬Empty(Prune1(𝒮, ⟨I_1, ..., I_{i-1}, I_c, I_{i+1}, ..., I_n⟩))};
  if C = ∅ then
    return ∅;
  else
    return [min_{I∈C} left(I), max_{I∈C} right(I)];
end
```

Figure 3. Implementing box(2)-consistency.

DEFINITION 11 (Box(2)-consistency pruning). *Let $\mathcal{S}$ be a system of interval constraints and let $\vec{I}$ be a box. The pruning operator associated with box(2)-consistency can be defined as*

$$\mathtt{Prune}_2(\mathcal{S}, \vec{I}) = \vec{I}'$$

where $\vec{I}'$ is the largest box in $\vec{I}$ such that $\mathcal{S}$ is box(2)- consistent in $\vec{I}$'.

5.3.3. *A simple implementation*

Once again, the pruning operator can be computed in many ways. Figure 3 presents a simple iterative algorithm close to our actual implementation. The algorithm is again a simple fixpoint algorithm that terminates when no further pruning can be obtained. The body of the loop applies a narrowing operation on each of the variables and produces a new box that is the intersection of all these narrowings. The narrowing function $\mathtt{Narrow}_2(\mathcal{S}, \vec{I}, i)$ returns the largest box $\vec{I}'$ in $\vec{I}$ such that $\mathcal{S}$ is box(2)-consistent in $\vec{I}'$ wrt i. The pruning operator of box(1)-consistency is used to compute this narrowing operator. Note that it would be easy to define any level of box-consistency at this point, since box$(k-1)$-consistency can be used to define box(k)-consistency in a generic way.

5.4. RELATED WORK

It is useful to relate these pruning operators to earlier work in constraint satisfaction. Projections, and approximations of projections, have been used extensively in the artificial intelligence community (under the name *consistency techniques*)

to solve discrete combinatorial problems [e.g., 16, 15]. They have been adapted to continuous problems [e.g., 5, 14] and used inside systems such as BNR- Prolog and CLP(BNR) [3, 21] and many systems since then. The techniques used in systems like BNR-Prolog and CLP(BNR) are weaker than box(1)-consistency, since they decompose all constraints into ternary constraints on distinct variables before applying a form of box(1)-consistency. They do not scale well on difficult problems. Box(1)-consistency was introduced in Benhamou [2]. It is related to the techniques presented in Hong and Stahl [10], which uses a similar idea for the splitting process. The consistency notions of L'Homme [14] are a weaker, and less effective, form of box(k)-consistency: it is obtained by decomposing all constraints into ternary constraints over distinct variables and by applying a form of box(k)-consistency on the resulting constraints.

6. The branch and prune algorithm revisited

We now reconsider the assumptions of Section 3. As mentioned in that section, our algorithm uses two interval extensions, the natural interval extension and the mean value interval extension, since distinct interval extensions may produce different prunings. In particular, the natural interval extension is generally better when far from a solution, while the mean value interval extension is more precise when close to a solution. This section reviews both extensions and reconsiders the overall branch-and-prune algorithm.

6.1. THE NATURAL INTERVAL EXTENSION

The simplest interval extension of a function is its natural interval extension. Informally speaking, it consists in replacing each number by the smallest interval enclosing it, each real variable by an interval variable and each real operation by its fixed interval extension. In the following, if f is a real function, $\hat{f}$ is its natural extension.

EXAMPLE 2 (Natural interval extension). *The natural interval extension of the function* $x_1(x_2 + x_3)$ *is the interval function* $X_1(X_2 + x_3)$.

The advantage of this extension is that it preserves how constraints are written and hence users of the system can choose constraint representations particularly appropriate for the problem at hand. It is useful to generalize the natural interval extension to a system of constraints.

DEFINITION 12 (Natural interval extension of a system). *Let* S *be a system of constraints*

$$S = \begin{cases} 0 = f_1(x_1, \ldots, x_n) \\ \ldots \\ 0 = f_n(x_1, \ldots, x_n) \end{cases}$$

The natural extension of S, denoted by $\hat{S}$, is the set of the interval constraints

$$S = \begin{cases} 0 = \widehat{f_1}(X_1, \ldots, X_n) \\ \ldots \\ 0 = \widehat{f_n}(X_1, \ldots, X_n) \end{cases}$$

The following result is easy to prove by induction.

PROPOSITION 2. *The natural interval extension of a function, a constraint, or a system of constraints is monotonic.*

6.2. THE MEAN VALUE INTERVAL EXTENSION

The second interval extension is based on the Taylor expansion around a point. This extension is an example of centered forms that are interval extensions introduced by Moore [17] and have been studied by many authors, since they have important properties. The mean value interval extension of a function is parametrized by the intervals for the variables in the function. It also assumes that the function has continuous partial derivatives. Given these assumptions, they key idea behind the extension is to apply a Taylor expansion of the function around the center of the box and to bound the rest of the series using the box.

DEFINITION 13 (Mean value interval extension). *Let $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$ and m_i be the center of I_i. The mean value interval extension of a function f in $\vec{I}$, denoted by $\tau(f, \vec{I})$, is the interval function*

$$\hat{f}(\overline{m_1}, \ldots, \overline{m_n}) \oplus \sum_{i=1}^{n} \frac{\widehat{\partial f}}{\partial x_i}(\vec{I})(X_i \ominus \overline{m_i}).$$

Let S be a system of constraints

$$S = \begin{cases} 0 = f_1(x_1, \ldots, x_n) \\ \ldots \\ 0 = f_n(x_1, \ldots, X_n) \end{cases}$$

The mean value interval extension of S in $\vec{I}$, denoted by $\tau(S, \vec{I})$, is the system of interval constraints

$$\tau(S, \vec{I}) = \begin{cases} 0 = \tau(f_1, \vec{I})(X_1, \ldots, X_n) \\ \ldots \\ 0 = \tau(f_n, \vec{I})(X_1, \ldots, X_n) \end{cases}$$

Note that the mean value interval extensions is defined in terms of natural extensions. The proof of the following proposition can be found in Caprani and Madsen [4].

```
function Search(S, Ī0)
begin
  Ī := Prune(Ŝ ∪ τc(S, Ī0), Ī0);
  if Empty(Ī) then
    return ∅
  else if Canonical(Ī) then
    return {Ī}
  else
    ⟨Ī1, Ī2⟩ := Split(Ī);
    return Search(S, Ī1) ∪ Search(S, Ī2);
end
```

Figure 4. The branch-and-prune algorithm revisited.

PROPOSITION 3. *The mean value interval extension of a function is a monotonic interval extension.*

It is interesting to note that box consistency on the mean value interval extension of a system of constraints is closely related to the Hansen-Sengupta operator [8], which is an improvement over Krawczyk's operator [13]. Hansen and Smith [9] also argue that these operators are more effective when the interval Jacobian of the system is diagonally dominant and they suggest conditioning the system $\mathcal{S}$. For the purpose of this paper, we simply assume that we have a conditioning operator $cond(S, \vec{I})$ and use the notation $\tau_c(S, \vec{I})$ to denote $\tau(cond(S, \vec{I}), \vec{I})$. See Kearfott [11, 12] for extensive coverage of conditioners.

6.3. THE BRANCH-AND-PRUNE ALGORITHM

We are now in position to reconsider our branch-and-prune algorithm. The new version, given in Figure 4, differs in two ways from the algorithm presented in Section 4. On the one hand, the algorithm receives as input a system of constraints (instead of a system of interval constraints). On the other hand, the operation `Prune` receives a system of interval constraints consisting of the natural interval extension and the conditioned mean value interval extensions of the original system.

6.4. EXISTENCE PROOF

We now describe how the algorithm proves the existence of a solution in a box. Let $\{f_1 = 0, \ldots, f_n = 0\}$ be a system of equations over variables $\{x_1, \ldots, x_n\}$, let

$\vec{I} = \langle I_1, \ldots, I_n \rangle$ be a box and define the intervals $I_i'(1 \leqslant i \leqslant n)$ as follows

$$I_i' = \left(\overline{m_i} \ominus \frac{1}{\frac{\widehat{\partial f_i}}{\partial x_i}(\vec{I})} \left[\sum_{j=1, j\neq 1}^{n} \frac{\widehat{\partial f_i}}{\partial x_j}(\vec{I})(I_j \ominus \overline{m_j}) \oplus \hat{f}_i(\overline{m_1}, \ldots, \overline{m_n}) \right] \right)$$

where $m_i = center(I_i)$. If

$$I_i' \subseteq I_i \quad (1 \leqslant i \leqslant n)$$

then there exists a zero in $\langle I_i', \ldots, I_n' \rangle$. A proof of this result can be found in Moore and Jones [19].

7. Experimental results

This section reports experimental results of the branch-and-prune algorithms. We compare the branch-and-prune algorithm with two instantiations of the pruning operator: $\texttt{Prune}_1$ to obtain the algorithm presented in Van Hentenryck et al. and $\texttt{Prune}_2$ to obtain a novel algorithm more effective for the transistor modelling problem.

The transistor modelling problem. The box(2)-consistency algorithm was applied to find all solutions of the transistor modelling problem. Branching was applied until a safe box or a box of width smaller than 10^{-8} was obtained. The algorithm returned a unique box

$$\begin{aligned}
x[1] &= 0.8999999 \quad + [0.48517e^{-7}, 0.566954e^{-7}] \\
x[2] &= 0.4499874 \quad + [0.6902216e^{-7}, 0.7493801e^{-7}] \\
x[3] &= 1.00000648 + [0.60195e^{-9}, 0.43303e^{-8}] \\
x[4] &= 2.00006854 + [0.5787e^{-10}, 0.319179e^{-8}] \\
x[5] &= 7.9999714 \quad + [0.3767867e^{-7}, 0.4259589e^{-7}] \\
x[6] &= 7.99969268 + [0.14994e^{-8}, 0.692803e^{-8}] \\
x[7] &= 5.00003127 + [0.338646e^{-8}, 0.848255e^{-8}] \\
x[8] &= 0.99998772 + [0.69887e^{-9}, 0.621097e^{-8}] \\
x[9] &= 2.00005248 + [0.47411e^{-9}, 0.649037e^{-8}]
\end{aligned}$$

in the original range $[0, 10]^9$, together with a proof that the box contains a solution. The algorithm performs only 118 branchings and takes 2359.80 seconds (roughly 40 minutes) on a Sun Ultra-2 running Solaris. This is of course a considerable improvement over the results of Ratschek and Rokne [23]. Interestingly, the box(1)-consistency algorithm performs 135099 branchings and takes 11841 seconds (roughly 3 hours and 20 minutes) on the same machine, still a considerable improvement over [23].

Table 1. Box(1)- versus box(2)-consistency on some benchmarks from continuation methods

Benchmarks	v	d	range	Box(1)-time	Box(1)-branch	Box(2)-time	Box(2)-branch
kin1	12	4608	$[-10^8, 10^8]$	15.1	307	46.20	15
kin2	8	256	$[-10^8, 10^8]$	186.1	3505	747.90	194
combustion	10	96	$[-10^8, 10^8]$	0.0	1	0.5	0
chemistry	5	108	$[0, 10^8]$	1.1	57	2.2	1

Benchmarks from continuation methods. It is worth comparing the box(1)- and box(2)-consistency algorithms on some standard benchmarks from continuation methods [25]. The box(1)-consistency algorithm compares well with continuation methods on these benchmarks [24]. Table 1 reports the results and gives, for each benchmark, the number of variables, the degree of the polynomial system, the initial range of the variables, and the CPU time and number of branchings of the box(1)- and box(2)-consistency algorithms. See [24, 25] for a description of the benchmarks. The intention is not to compare the two algorithms systematically but rather to make readers aware that neither of two algorithms is really superior. As can be seen, the box(2)-consistency algorithm always performs less branchings (as should be expected) but is always slower than the box(1)-consistency algorithm. On these problems, box(1)-consistency seems to give a better tradeoff between pruning and pruning time. A fundamental topic for future research is thus to determine when box(2)-consistency is more effective than box(1)- consistency and, more generally, to characterize the class of nonlinear problems for which these techniques are effective.

8. Conclusions

This paper reconsidered the transistor modelling problem from Ebers and Moll [6], which consists of nine nonlinear equations and is challenging both for local and global methods. The problem was tackled by a novel branch-and-prune algorithm combining techniques from interval methods and constraint satisfaction. In particular, the algorithm enforces a local consistency condition called box(2)-consistency that strengthens the notion of box(1)-consistency introduced in [24]. The algorithm was applied to find all solutions to the transistor modelling problem and returned a unique safe box in the range $[0, 10]^9$ in about 40 minutes, performing only 118 branchings. The paper also indicated that box(2)-consistency may be too strong a local condition for many problems, since it is slower than box(1)-consistency on benchmarks from continuation methods [25]. An interesting avenue of research is to characterize more formally the class of applications for which box(1)- and box(2)-consistency are effective pruning techniques.

Acknowledgments

Special thanks to Christian Bliek for bringing the transistor modelling problem to our attention and to Yves Deville for many interesting discussions. This research was partly supported by the Office of Naval Research under grant N00014-91-J-4052 ARPA order 8225, the National Science Foundation under grant numbers CCR-9357704, a NSF National Young Investigator Award.

References

1. G. Alefeld and J. Herzberger (1983), *Introduction to Interval Computations*. New York: Academic Press.
2. F. Benhamou, D. McAllister, and P. Van Hentenryck (1994), CLP (Intervals) revisited, in *Proceedings of the International Symposium on Logic Programming (ILPS-94)*, pages 124–138. Ithaca, NY.
3. F. Benhamou and W. Older (1997), Applying interval arithmetic to real, integer and Boolean constraints, *Journal of Logic Programming* 32(1): 1–24.
4. O. Caprani and K. Madsen (1980), Mean value forms in interval analysis, *Computing* 25: 147–154.
5. J.G. Cleary (1987), Logical arithmetic, *Future Generation Computing Systems* 2(2): 125–149.
6. J.J. Ebers and J.L. Moll (1954), Large-scale behaviour of junction transistors, *IEE Proc.* 42: 1761–1772.
7. E.R. Hansen and R.I. Greenberg (1983), An interval Newton method, *Appl. Math. Comput.* 12: 89–98.
8. E.R. Hansen and S. Sengupta (1981), Bounding solutions of systems of equations using interval analysis, *BIT* 21: 203–211.
9. E.R. Hansen and R.R. Smith (1967), Interval arithmetic in matrix computation: Part II, *SIAM Journal on Numerical Analysis* 4: 1–9.
10. H. Hong and V. Stahl (1994), Safe starting regions by fixed points and tightening, *Computing*, 53(3-4): 323–335.
11. R.B. Kearfott (1990), Preconditioners for the interval Gauss-Seidel method, *SIAM Journal of Numerical Analysis* 27: 804– 822.
12. R.B. Kearfott (1991), A review of preconditioners for the interval Gauss-Seidel method, *Interval Computations* 1: 59–85.
13. R. Krawczyk (1969), Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, *Computing* 4: 187–201.
14. O. L'Homme (1993), Consistency techniques for numerical constraint satisfaction problems, in: *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*. Chamberry, France.
15. A.K. Mackworth (1977), Consistency in networks of relations, *Artificial Intelligence* 8(1): 99–118.
16. U. Montanari (1974), Networks of constraints: fundamental properties and applications to picture processing, *Information Science* 7(2): 95–132.
17. R.E. Moore (1966), *Interval Analysis*. Englewood Cliffs, NJ: Prentice-Hall.
18. R.E. Moore (1979), *Methods and Applications of Interval Analysis*. SIAM Publ.
19. R.E. Moore and S.T. Jones (1977), Safe starting regions for iterative methods, *SIAM Journal on Numerical Analysis* 14: 1051– 1065.
20. A. Neumaier (1990), *Interval Methods for Systems of Equations*. PHI Series in Computer Science. Cambridge: Cambridge University Press.

21. W. Older and A. Vellino (1993), Constraint arithmetic on real intervals, in *Constraint Logic Programming: Selected Research*. Cambridge, Mass.: The MIT Press.
22. H. Ratschek and J. Rokne (1988), *New Computer Methods for Global Optimization*. Chichester: Ellis Horwood Ltd.
23. H. Ratschek and J. Rokne (1993), Experiments using interval analysis for solving a circuit design problem, *Journal of Global Optimization* 3: 501–518.
24. P. Van Hentenryck, D. McAllister, and D. Kapur (1997), Solving polynomial systems using a branch and prune approach, *SIAM Journal on Numerical Analysis* 34(2): 797–827.
25. J. Verschelde, P. Verlinden, and R. Cools (1994), Homotopies exploiting Newton polytopes for solving sparse polynomial systems, *SIAM Journal on Numerical Analysis* 31(3): 915–930.