A Constraint Satisfaction Approach to a Circuit Design Problem

Jean-Francois Puget
and Pascal Van Hentenryck

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-34
December 1996

A Constraint Satisfaction Approach to a Circuit Design Problem

Jean-Francois Puget
Ilog SA,
9, rue de Verdun
F-94253 Gentilly (France)
puget@ilog.fr

Pascal Van Hentenryck
Brown University
Box 1910
Providence, RI 02912
pvh@cs.brown.edu

Abstract

A classical circuit design problem from Ebers and Moll [5] features a system of 9 nonlinear equations in 9 variables which is very challenging both for local and global methods. This system was solved globally using an interval method by Ratschek and Rokne [21] in the box $[0, 10]^9$ and they state

> *Our strategies are far from being perfect. They were however successful, despite enormous costs. The costs were so high that the computation for this problem is certainly not competitive, but 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.*

This paper presents a novel branch and prune algorithm which obtains a unique safe box to 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.

Keywords: Global zero search, electrical circuit, transistor modeling, interval methods, branch and prune, constraint satisfaction.

1 Introduction

The transistor modeling problem of Ebers and Moll [5] is the system of nonlinear equations

$$\begin{cases} (1 - x_1x_2)x_3 \; [\; exp(x_5(g_{1k} - g_{3k}x_7 10^{-3} - g_{5k}x_8 1e-3)) - 1 \;] - g_{5k} + g_{4k}x_2 = 0 \;\; (1 \le k \le 4) \\ (1 - x_1x_2)x_3 \; [\; exp(x_6(g_{1k} - g_{2k} - g_{3k}x_7 10^{-3} + g_{4k}x_9 10^{-3})) - 1 \;] - g_{5k}x_1 + g_{4k} = 0 \;\; (1 \le k \le 4) \\ x_1x_3 - x_2x_4 = 0; \end{cases}$$

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 [21] summarize various attempts to find a solution to this problem using local methods and these descriptions will not be repeated here. It suffices to say that successful attempts require very elaborate procedures, sometimes combining several globally convergent algorithms.

Ratschek and Rokne in the same paper [21] 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 extremely high however. They state in their introduction

> *Our strategies are far from being perfect. They were however successful, despite enormous costs. The costs were so high that the computation for this problem is certainly not competitive, but at this time, we know no other method which has been applied to this circuit design problem and which has led to the same garanteed result of locating exactly one solution in this huge domain, completed with a reliable error estimate.*

and in their conclusion

> *Although the computation was successful it was extremely costly in terms of machine cycles. It is therefore recommended that such methods for solving unstable systems should be further investigated with the aim of reducing the computations. These methods would be based on improved selection criteria for the subdivision process.*

No computation times were given in [21], since this was not the primary aim of the paper. However, a preliminary draft of the paper indicated that the overall process took over 14 months using a network of 30 Sun Sparc 1 workstations.

In this paper, we present a novel interval algorithm to isolate all safe boxes (i.e., all boxes which contain at least one solution) to nonlinear equation systems and we show that the algorithm isolates a single safe box for the circuit design problem within reasonable computation times.[1] The main novelty of the procedure is not in an improved selection criteria for the subdivision process but rather in the way constraints are used to prune the search space. The pruning techniques, some of which were presented in [22] 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 rest of this paper is organized as follows. Section 2 gives the necessary background in interval analysis. Section 3 presents the branch and prune algorithm for isolating the solutions to a nonlinear system. Section 4 reports the experimental results and Section 5 concludes the paper.

[1] Proving that the safe box contains a unique solution requires some experimental trial and error work because of the nature of these proofs. See Section 6 of [21] for a discussion on this issue.

2 Interval Analysis

In this section, we review some basic concepts needed for this paper, including interval arithmetic and the representation of constraints. More information on interval arithmetic can be found in many places (e.g., [1, 7, 6, 15, 16, 18, 20]). Our definitions are slightly non-standard.

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

$$\{r \in \Re \mid l \leq r \leq u\}.$$

The set of intervals is denoted by $\mathcal{I}$ and is ordered by set inclusion.[2]

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 $\mathcal{I} \rightarrow \mathcal{I}$) by the letters F, G, all possibly subscripted. We use l^+ (resp. l^-) to denote the smallest (resp. 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

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

and the set of solutions of a system $\mathcal{S}$ in a box $\vec{I_0}$ is denoted by $RSol(\mathcal{S}, \vec{I_0})$, i.e.,

$$RSol(\mathcal{S}, \vec{I_0}) = \{\vec{r} \in \vec{I_0} \mid \forall c \in \mathcal{S} : c(\vec{r})\}.$$

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

Definition 3 [Set Extensions] Consider a set $S \subseteq \Re^n$, a function $f : \Re^n \rightarrow \Re$, and a relation $c \subseteq \Re^n$. Then, the set extensions of f and c are defined as

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

and

$$c(S) \Leftrightarrow \exists \vec{r} \in S \ c(\vec{r}).$$

[2] Our intervals are usually called floating-point intervals in the literature.

Definition 4 [Interval Extensions] An interval function $F : \mathcal{I}^n \to \mathcal{I}$ is an *interval extension* of $f : \Re^n \to \Re$ in $\vec{I_0}$ if

$$\forall \vec{I} \subseteq \vec{I_0} : f(\vec{I}) \subseteq F(\vec{I}).$$

An interval relation $C \subseteq \mathcal{I}^n$ is an *interval extension* of a relation $c \subseteq \Re^n$ in $\vec{I_0}$ if

$$\forall \vec{I} \subseteq \vec{I_0} : c(\vec{I}) \Rightarrow C(\vec{I}).$$

A system of interval constraints $\breve{\mathcal{S}}$ is an *interval extension* of a system of constraints $\mathcal{S}$ in $\vec{I_0}$ if

$$\forall \vec{I} \subseteq \vec{I_0} : \mathcal{S}(\vec{I}) \Rightarrow \breve{\mathcal{S}}(\vec{I})$$

where

$$\mathcal{S}(\vec{I}) \;\Leftrightarrow\; RSol(\mathcal{S}, \vec{I}) \neq \emptyset$$

and

$$\breve{\mathcal{S}}(\vec{I}) \;\Leftrightarrow\; \forall C \in \breve{\mathcal{S}} : C(\vec{I}).$$

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. The interval relation $\approx$ defined as

$$I_1 \approx I_2 \;\Leftrightarrow\; (I_1 \cap I_2 \neq \emptyset)$$

is an interval extension of the equality relation on 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 5 [Monotonic Interval Extensions] An interval function $F : \mathcal{I}^n \to \mathcal{I}$ is *monotonic* in $\vec{I_0}$ if

$$\forall \vec{I_1}, \vec{I_2} \subseteq \vec{I_0} : \vec{I_1} \subseteq \vec{I_2} \Rightarrow F(\vec{I_1}) \subseteq F(\vec{I_2}).$$

An interval relation $C \subseteq \mathcal{I}^n$ is monotonic in $\vec{I_0}$ if

$$\forall \vec{I_1}, \vec{I_2} \subseteq \vec{I_0} : \vec{I_1} \subseteq \vec{I_2} \Rightarrow [C(\vec{I_1}) \Rightarrow C(\vec{I_2})].$$

A system of interval constraints $\breve{\mathcal{S}}$ is *monotonic* in $\vec{I_0}$ if

$$\forall \vec{I_1}, \vec{I_2} \subseteq \vec{I_0} : \vec{I_1} \subseteq \vec{I_2} \Rightarrow [\breve{\mathcal{S}}(\vec{I_1}) \Rightarrow \breve{\mathcal{S}}(\vec{I_2})].$$

It is important to stress that a real function (resp. relation) 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 and relations (for instance, the interval extension of $+$ is defined by $\oplus$). In addition, we overload the real symbols and use them for their interval extensions. Finally, we denote relations by the letter c possibly subscripted, interval relations by the letter C possibly subscripted. Note that constraints and relations are used as synonyms in this paper.

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 constraints are written may have an impact on the behaviour 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. For the remaining sections, we assume that real variables in constraints are taken from a finite (but arbitrary large) set $\{x_1, \ldots, x_n\}$. Similar conventions apply to interval functions and constraints. Interval variables are taken from a finite (but arbitrary large) set $\{X_1, \ldots, X_n\}$.

3 The Branch and Prune Algorithm

This section presents a high-level overview of our branch & prune algorithm to find all isolated solutions of a system of nonlinear equations. The algorithm associates an interval with each variable and iterates two steps:

(i) Pruning: using the constraints to reduce the intervals associated with the variables;

(ii) Branching: splitting an interval in two parts and exploring recursively the two new subproblems.

The pruning step applies a local consistency condition at each node of the search tree. A previous version of our algorithm [22] uses a condition called *box consistency*. The algorithm presented here enforces a stronger local condition called *bound consistency*.

The rest of this section is organized as follows. Section 3.1 reviews the concept of box consistency, Section 3.2 discusses interval extensions for box consistency, and Section 3.3 introduces the concept of bound consistency. Section 3 describes the branch and prune algorithm while Section 3.5 presents a simple way to prove existence of a solution in a box.

3.1 Box Consistency

Box consistency [2] is an approximation of arc consistency, a notion well-known in artificial intelligence [13] which states a simple local condition on a constraint c and the set of possible values for each of its variables, say $D_1, \ldots, D_n$. Informally speaking, a constraint c is arc consistent if none of the D_i can be reduced by using projections of c.

Definition 6 [Projection Constraint] A projection constraint $\langle c, i\rangle$ is a pair of a constraint c and an index i $(1 \leq i \leq n)$.

Example 2 Consider the constraint $x_1^2 + x_2^2 = 1$. Both $\langle x_1^2 + x_2^2 = 1\,,\,1\rangle$ and $\langle x_1^2 + x_2^2 = 1\,,\,2\rangle$ are projection constraints.

Definition 7 [Arc Consistency] A projection constraint $\langle c, i \rangle$ is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i \subseteq \{r_i \mid \exists r_1 \in D_1, \ldots, \exists r_{i-1} \in D_{i-1}, \exists r_{i+1} \in D_{i+1}, \ldots, \exists r_n \in D_n : c(r_1, \ldots, r_n)\}$. A constraint c is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$ if each of its projections is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$. A system of constraints $\mathcal{S}$ is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$ if each constraint in $\mathcal{S}$ is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$.

Example 3 Let c be the constraint $x_1^2 + x_2^2 = 1$. c is arc-consistent wrt $\langle\ [-1, 1]\ ,\ [-1, 1]\ \rangle$ but is not arc-consistent wrt $\langle\ [-1, 1]\ ,\ [-2, 2]\ \rangle$ since, for instance, there is no value r_1 for x_1 in $[-1, 1]$ such that $r_1^2 + 2^2 = 1$.

Given some initial domains $\langle D_1^0, \ldots, D_n^0 \rangle$, an arc consistency algorithm computes the largest domains $\langle D_1, \ldots, D_n \rangle$ included in $\langle D_1^0, \ldots, D_n^0 \rangle$ such that all constraints are arc-consistent. These domains always exist and are unique. Enforcing arc consistency is often effective on discrete combinatorial problems. However, it cannot be computed in general when working with real numbers and polynomial constraints. Moreover, simple approximations to arc consistency taking into account numerical accuracy are very expensive to compute (the exact complexity is an open problem). For instance, a simple approximation of arc consistency consists in working with intervals and approximating the set computed by arc consistency to return an interval, i.e.,

$$I_i \subseteq box\{\{\ r_i \mid \exists r_1 \in I_1, \ldots, \exists r_{i-1} \in I_{i-1}, \ldots, \exists r_{i+1} \in I_{i+1}, \exists r_n \in I_n : c(r_1, \ldots, r_n)\ \}\}.$$

This condition, used in systems like [19, 3], is easily enforced on simple constraints such as

$$x_1 = x_2 + x_3, \quad x_1 = x_2 - x_3, \quad x_1 = x_2 \times x_3$$

but it is also computationally very expensive for complex constraints with multiple occurrences of the same variables. Moreover, decomposing complex constraints into simple constraints entails a substantial loss in pruning, making this approach not practical on many applications. See [2] for experimental results on this approach and their comparison with the approach presented in this paper.

The notion of box consistency introduced in [2] is a coarser approximation of arc consistency which provides a much better trade-off between efficiency and pruning. It consists in replacing the existential quantification in the above condition by the evaluation of an interval extension of the constraint on the intervals of the existential variables. Since there are many interval extensions for a single constraint, we define box consistency in terms of interval constraints.

Definition 8 [Interval Projection Constraint] An interval projection constraint $\langle C, i \rangle$ is the association of an interval constraint C and of an index i $(1 \leq i \leq n)$. Interval projection constraints are denoted by the letter P, possibly subscripted.

Definition 9 [Box Consistency] An interval projection constraint $\langle C, i \rangle$ is *box-consistent* in the intervals $\vec{I} = \langle I_1, \ldots, I_n \rangle$ if

$$I_i = \Box\{\ r_i\ \in\ I_i \mid C(I_1, \ldots, I_{i-1}, \overline{r_i}, I_{i+1}, \ldots, I_n)\}$$

An interval constraint is *box-consistent* in $\vec{I}$ if its projections are box-consistent in $\vec{I}$. A system of interval constraints is *box-consistent* in $\vec{I}$ if its interval constraints are box-consistent in $\vec{I}$.

Proposition 1 Let $\langle C, i \rangle$ be an interval projection constraint and $\vec{I}$ be a box such that $l_i = \mathit{left}(I_i)$ and $u_i = \mathit{right}(I_i)$. If

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

and

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

then $\langle C, i \rangle$ is *box-consistent* in $\vec{I}$.

Proof Let $S = \{ \ r_i \ \in \ I_i \mid C(I_1, ..., I_{i-1}, \overline{r_i}, I_{i+1}, ..., I_n)\}$. The result is obvious when $l_i = u_i$. For the general case, consider any r in $]l_i, l_i^+[$. Since $\overline{r} = [l_i, l_i^+]$, it follows that $r \in S$. Similarly, consider any r in $]u_i^-, u_i[$. Since $\overline{r} = [u_i^-, u_i]$, it follows that $r \in S$. As a consequence, $\Box S = [l_i, u_i]$ and the result follows. $\Box$

Intuitively speaking, the above condition states that the ith interval cannot be pruned further using the unary interval constraint obtained by replacing all variables but X_i by their intervals, since the boundaries satisfy the unary constraint. It should be clear that box consistency is an approximation of arc consistency.[3] The difference between arc consistency and box consistency appears essentially when there are multiple occurrences of the same variable.

Example 4 Consider the constraint

$$x_1 + x_2 - x_1 = 0.$$

The constraint is not arc-consistent in $\langle[-1, 1], [-1, 1]\rangle$ since there is no value r_1 for x_1 which satisfies $r_1 + 1 - r_1 = 0$. On the other hand, the interval constraint

$$X_1 \oplus X_2 \ominus X_1 \approx 0$$

is box-consistent, since

$$([-1, 1] \oplus [-1, -1^+] \ominus [-1, 1]) \cap [0, 0] \ \neq \ \emptyset$$

and

$$([-1, 1] \oplus [1^-, 1] \ominus [-1, 1]) \cap [0, 0] \ \neq \ \emptyset.$$

3.2 Interval Extensions for Box Consistency

Box consistency strongly depends on the interval extensions chosen for the constraints and different interval extensions can produce different (often incomparable) tradeoffs between pruning and computational complexity. Our algorithm uses two extensions: the natural interval extension and the mean value interval extension.

[3] It is interesting to note that this definition is also related to the theorem of Miranda [18]. In this case, box consistency can be seen as replacing universally quantified variables by their intervals.

3.2.1 The Natural Interval Extension

The simplest interval extension of a function (resp. constraint) 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, each real operation by its fixed interval extension and each real relation by its fixed interval extension. In the following, if f (resp. c) is a real function (resp. constraint), we denote its natural extension by $\widehat{f}$ (resp. $\widehat{c}$).

Example 5 [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 natural interval extension of the constraint $x_1(x_2+x_3) = 0$ is the interval constraint $X_1(X_2 + X_3) = \overline{0}$.

The advantage of this extension is that it preserves the way 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 10 [Natural Interval Extension of a System] Let $\mathcal{S}$ be a system of constraints $\{c_1, \ldots, c_n\}$. The *natural extension* of $\mathcal{S}$, denoted by $\widehat{\mathcal{S}}$, is the set of the interval constraints $\{\widehat{c_1}, \ldots, \widehat{c_n}\}$.

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.

3.2.2 The Mean Interval Extension

The second interval extension is based on the Taylor expansion around a point. This extension is an example of centered forms which are interval extensions introduced by Moore [15] and studied by many authors, since they have important properties. The mean value interval extension of a constraint is parametrized by the intervals for the variables in the constraint. It also assumes that the constraint which it is applied to is of the form $f = 0$ where f denotes a function which has continuous partial derivatives. Given these assumptions, the 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 11 [Mean Value Interval Extensions] 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

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

The mean value interval extension of a constraint $f_1\ \omega\ f_2$ in $\vec{I}$, denoted by $\tau(f_1\ \omega\ f_2, \vec{I})$, is the interval constraint

$$\tau(f_1, \vec{I})\ \widehat{\omega}\ \tau(f_2, \vec{I}).$$

The mean value interval extension of a system $\mathcal{S} = \{c_1, \ldots, c_n\}$ in $\vec{I}$, denoted by $\tau(\mathcal{S}, \vec{I})$, is the system

$$\{\tau(c_1, \vec{I}), \ldots, \tau(c_n, \vec{I})\}.$$

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

Proposition 3 The mean value interval extension of a function, a constraint, or a system of constraints 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's operator [7], which is an improvement over Krawczyk's operator [11]. Hansen and Smith also argue that these operators are more effective when the interval Jacobian of the system is diagonally dominant [8] and they suggest to condition the system $\mathcal{S}$. For the purpose of this paper, it is sufficient to abstract the notion of conditioning by the following definition.

Definition 12 [Conditioning] Let $\mathcal{S}$ be a system of constraints and let $\vec{I}$ be a box. A *conditioning* of $\mathcal{S}$ in $\vec{I}$ is a system of constraints $\mathcal{S}'$ satisfying $RSol(\mathcal{S}, \vec{I}) = RSol(\mathcal{S}', \vec{I})$.

The conditioning used in our algorithm for a system $\mathcal{S}$ in a box $\vec{I}$ is denoted by $cond(\mathcal{S}, \vec{I})$ in the following and we use the notation $\tau_c(\mathcal{S}, \vec{I})$ to denote $\tau(cond(\mathcal{S}, \vec{I}), \vec{I})$. See [9, 10] for an extensive coverage of conditioners.

3.3 Bound Consistency

Box consistency has been shown to be effective for solving a variety of nonlinear applications [22]. For some applications however, and for the transistor modeling problem in particular, a better efficiency can be obtained by using a stronger local consistency condition that we call *bound consistency.* Bound consistency is related to the notions of path-consistency [14] and to the consistency notions presented in [12]. It is is defined in terms of box consistency or, more precisely, in terms of whether a system can be made box-consistent.

Definition 13 [Box-satisfiability] A system of interval constraints $\breve{\mathcal{S}}$ is *box-satisfiable* in $\vec{I_0}$, denoted $BoxSat(\breve{\mathcal{S}}, \vec{I_0})$, if there exists a non-empty box $\vec{I} \subseteq \vec{I_0}$ such that $\breve{\mathcal{S}}$ is box-consistent in $\vec{I}$.

Bound consistency then consists in checking if the bounds of each variable are box-satisfiable.

Definition 14 [Bound Consistency] Let $\breve{\mathcal{S}}$ be a system of interval constraints and $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$ with I_j be $[l_j, u_j]$. Variable x_i is *bound-consistent* in $\breve{\mathcal{S}}$ and $\vec{I}$ if

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

when $l_i \neq u_i$ and if

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

The system $\breve{\mathcal{S}}$ is *bound-consistent* in $\vec{I}$ if each variable x_i $(1 \leq i \leq n)$ is bound-consistent in $\breve{\mathcal{S}}$ and in $\vec{I}$.

It can be shown that bound consistency implies box consistency.

Proposition 4 Let $\breve{\mathcal{S}}$ be a system of monotonic interval constraints. If $\breve{\mathcal{S}}$ is bound-consistent in $\vec{I}$, then $\breve{\mathcal{S}}$ is box-consistent in $\vec{I}$.

Proof Assume for simplicity that $\vec{I} = \langle I_1, \ldots, I_n \rangle$ with $I_i = [l_i, u_i]$ and $l_i \neq u_i$. The proof is similar otherwise. Since $\breve{\mathcal{S}}$ is bound-consistent in $\vec{I}$, variable X_i is bound-consistent in $\breve{\mathcal{S}}$ and $\vec{I}$ $(1 \leq i \leq n)$. Thus,

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

or, more explicitly,

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

It follows by Proposition 1 that, for all $C \in \breve{\mathcal{S}}$,

$$C(\vec{I'})$$

and, by monotonicity of C, that

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

The proof of

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

is similar. The result follows from the definition of box consistency. $\Box$

3.4 The Branch and Prune Algorithm

We are now in position to present our branch and prune algorithm. The algorithm applies, at each node of the search tree, a pruning step to reduce the intervals of the variables so that

- the constraint system is bound-consistent in the reduced intervals for the natural interval extension and the mean value interval extension;
- no solution is removed.

Specification 1 [Pruning] Let $\mathcal{S}$ be a system of constraints and let $\vec{I_0}$ be a non-empty box. Function `PRUNE(`$\mathcal{S}$`,`$\vec{I_0}$`)` returns a box $\vec{I}$ satisfying the following conditions

1. $\vec{I} \subseteq \vec{I_0}$,
2. $RSol(\mathcal{S}, \vec{I_0}) = RSol(\mathcal{S}, \vec{I})$,
3. $\widehat{\mathcal{S}}$ is bound-consistent in $\vec{I}$,
4. $\tau_c(\mathcal{S}, \vec{I_0})$ is bound-consistent in $\vec{I}$.

```
function BranchAndPrune(𝒮: Set of Constraint; I⃗₀ : ℐⁿ): Set of ℐⁿ;
begin
   I⃗ := PRUNE(𝒮, I⃗₀);
   if ¬ IsEmpty(I⃗) then
      if IsSmallEnough(I⃗) then
         return {I⃗}
      else
         ⟨I⃗₁, I⃗₂⟩ := BRANCH(I⃗);
         return BranchAndPrune(𝒮, I⃗₁) ∪ BranchAndPrune(𝒮, I⃗₂)
      endif
   else
      return ∅
   endif
end
```

Figure 1: The Branch and Prune Algorithm

The high-level description of the branch and prune algorithm is described in Figure 1. The algorithm applies operation PRUNE on the initial box. If the resulting box is empty (i.e., one of its components is empty), then there is no solution. If the resulting box is small enough (specified by the desired accuracy in solutions), then it is included in the result. The function BRANCH splits the box into two subboxes along one dimension (variable). The choice of the variable to branch next is important and, in our experimental results, we simply assume that the variable with the largest interval is selected. See also [22] for other pragmatic considerations in implementing the ideal model described here.

3.5 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 $(1 \leq i \leq n)$ as follows

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

where $m_i = center(I_i)$. If

$$I'_i \subseteq I_i \qquad (1 \leq i \leq n))$$

then there exists a zero in $\langle I'_1, \ldots, I'_n\rangle$. A proof of this result can be found in [17].

Benchmarks	v	d	range	Box-Time	Box-Branch.	Bound-Time	Bound-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

Table 1: Box Versus Bound Consistency on some Benchmarks from Continuation Methods.

4 Experimental Results

This section reports experimental results of the branch and prune algorithm. In the result, the bound-consistency algorithm means the branch and prune algorithm presented in the previous section, while the box-consistency algorithm is the same algorithm, except that the pruning step applies box-consistency instead of bound-consistency. The box-consistency algorithm was presented in [22].

The Transistor Modeling Problem The bound-consistency algorithm was applied to find all solutions of the transistor modeling 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

```
x[1] = 0.8999999      + [0.485178e-7 , 0.566954e-7]
x[2] = 0.4499874      + [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      + [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]
```

in the original range $[0, 10]^9$, together with a proof that the box contains a solution. The algorithm only performs 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 [21]. Interestingly, the box-consistency algorithm performs 135099 branchings and takes 11841 seconds (roughly 3 hours and 20 minutes) on the same machine. This is of course still a considerable improvement over [21].

Benchmarks from Continuation Methods It is worth comparing the box-consistency and bound-consistency algorithms on some standard benchmarks from continuation methods [23]. The box-consistency algorithm was shown to compare well with continuation methods on these benchmarks [22]. 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-consistency and bound-consistency algorithms. See [23, 22] for a description of the benchmarks. The intention is not to compare the two algorithms systematically but rather to make readers aware that none of two algorithms is really superior. As can be seen, the bound-consistency algorithm performs always less branchings (as should be expected) but is always slower than the box-consistency algorithm. On these problems, box consistency seems a better tradeoff between pruning and pruning time. A fundamental topic for future research is thus to determine when bound consistency is more effective than box consistency and, more generally, to characterize the class of nonlinear problems for which these techniques are effective.

5 Conclusion

In this paper, the transistor modeling problem from Ebers and Moll [5] was reconsidered. This problem consists of 9 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 bound consistency which strengthens the notion of box consistency introduced in [22]. The algorithm was applied to find all solutions to the transistor modeling problem and it returned a unique safe box in the range $[0, 10]^9$ in about 40 minutes, performing only 118 branchings. The paper also indicated that bound consistency may be too strong a local condition for many problems, since it is slower than box-consistency on benchmarks from continuation methods [23]. An interesting avenue of research is to characterize more formally the class of applications for which bound consistency and box consistency are effective pruning techniques.

Acknowledgments

Special thanks to Christian Bliek for bringing the transistor modeling 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. *Introduction to Interval Computations.* Academic Press, New York, NY, 1983.

[2] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of the International Symposium on Logic Programming (ILPS-94)*, pages 124–138, Ithaca, NY, November 1994.

[3] F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1995. To appear.

[4] O. Caprani and K. Madsen. Mean Value Forms in Interval Analysis. *Computing*, 25:147–154, 1980.

[5] J.J. Ebers and J.L. Moll. Large-Scale Behaviour of Junction Transistors. *IEE Proc.*, 42:1761–1772, 1954.

[6] E.R. Hansen and R.I. Greenberg. An Interval Newton Method. *Appl. Math. Comput.*, 12:89–98, 1983.

[7] E.R. Hansen and S. Sengupta. Bounding Solutions of Systems of Equations Using Interval Analysis. *BIT*, 21:203–211, 1981.

[8] E.R. Hansen and R.R. Smith. Interval Arithmetic in Matrix Computation: Part II. *SIAM Journal on Numerical Analysis*, 4:1–9, 1967.

[9] R.B. Kearfott. Preconditioners for the Interval Gauss-Seidel Method. *SIAM Journal of Numerical Analysis*, 27, 1990.

[10] R.B. Kearfott. A Review of Preconditioners for the Interval Gauss-Seidel Method. *Interval Computations 1*, 1:59–85, 1991.

[11] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[12] O. Lhomme. Consistency Techniques for Numerical Constraint satisfaction Problems. In *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*, Chamberry, France, August 1993.

[13] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[14] U. Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95–132, 1974.

[15] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[16] R.E. Moore. *Methods and Applications of Interval Analysis.* SIAM Publ., 1979.

[17] R.E. Moore and S.T. Jones. Safe Starting Regions for Iterative Methods. *SIAM Journal on Numerical Analysis*, 14:1051–1065, 1977.

[18] A. Neumaier. *Interval Methods for Systems of Equations.* PHI Series in Computer Science. Cambridge University Press, Cambridge, 1990.

[19] W. Older and A. Vellino. Extending Prolog with Constraint Arithmetics on Real Intervals. In *Canadian Conference on Computer & Electrical Engineering*, Ottawa, 1990.

[20] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Limited, Chichester, 1988.

[21] H. Ratschek and J. Rokne. Experiments Using Interval Analysis for Solving a Circuit Design Problem. *Journal of Global Optimization*, 3:501–518, 1993.

[22] P. Van Hentenryck, D. McAllister, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 34(2), 1997.

[23] J Verschelde, P. Verlinden, and R. Cools. Homotopies Exploiting Newton Polytopes For Solving Sparse Polynomial Systems. *SIAM Journal on Numerical Analysis*, 31(3):915–930, 1994.