Solving Polynomial Systems Using a Branch and Prune Approach

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

D. McAllester
MIT AI Lab
Technology Square, 545
Cambridge, USA
dam@ai.mit.edu

D. Kapur
SUNY at Albany
Dep. of Computer Science
Albany, NY-12222
kapur@cs.albany.edu

Abstract

This paper presents Newton, a branch & prune algorithm to find all isolated solutions of a system of polynomial constraints. Newton can be characterized as a global search method which uses intervals for numerical correctness and for pruning the search space early. The pruning in Newton consists in enforcing at each node of the search tree a unique local consistency condition, called box-consistency, which approximates the notion of arc-consistency well-known in artificial intelligence. Box-consistency is parametrized by an interval extension of the constraint and can be instantiated to produce the Hansen-Segupta's narrowing operator (used in interval methods) as well as new operators which are more effective when the computation is far from a solution. Newton has been evaluated on a variety of benchmarks from kinematics, chemistry, combustion, economics, and mechanics. On these benchmarks, it outperforms the interval methods we are aware of and compares well with *state-of-the-art* continuation methods. Limitations of Newton (e.g., a sensitivity to the size of the initial intervals on some problems) are also discussed. Of particular interest is the mathematical and programming simplicity of the method.

AMS subject Classification: 65H10, 65G10
Keywords: System of Equations, Global Methods, Interval and Finite Analysis

1 Introduction

Many applications in science and engineering (e.g., chemistry, robotics, economics, mechanics) require finding all isolated solutions to a system of polynomial constraints over real numbers. This problem is difficult due to its inherent computational complexity (i.e., it is NP-hard) and due to the numerical issues involved to guarantee correctness (i.e., finding all solutions) and to ensure termination. Several interesting methods have been proposed in the past for this task, including two fundamentally different methods: interval methods (e.g., [4, 5, 7, 8, 11, 13, 14, 15, 19, 25, 29]) and continuation methods (e.g., [24, 35]). Continuation methods have been shown to be effective for problems for which the total degree is not too high, since the number of paths explored depends on the estimation of the number of solutions. Interval methods are generally robust but tend to be slow.

The purpose of this paper is to propose and to study a novel interval algorithm implemented in a system called Newton. From a user standpoint, Newton receives as input a system of polynomial constraints over, say, variables $x_1, \ldots, x_n$ and a box, i.e., an interval tuple $\langle I_1, \ldots, I_n \rangle$ specifying the initial range of these variables; it returns a set of small boxes of specified accuracy containing all solutions with typically one solution per box returned.

Operationally, `Newton` uses a branch & prune algorithm which was inspired by the traditional branch and bound approach used to solve combinatorial optimization problems. `Newton` uses intervals to address the two fundamental problems listed above. Numerical reliability is obtained by evaluating functions over intervals using outward rounding (as in interval methods). The complexity issue is addressed by using constraints to reduce the intervals early in the search. The pruning in `Newton` is achieved by enforcing a unique local consistency condition, called box-consistency, at each node of the search tree. Box-consistency is an approximation of arc-consistency, a notion well-known in artificial intelligence [16, 18] and used to solve discrete combinatorial problems in several systems (e.g.,, [31, 32]). Box-consistency is parametrized by an interval extension operator for the constraint and can be instantiated to produce various narrowing operators. In particular, box-consistency on the Taylor extension of the constraint produces a generalization of the Hansen-Segupta operator [8], well-known in interval methods. In addition, box-consistency on the natural extension produces narrowing operators which are more effective when the algorithm is not near a solution. `Newton` has the following properties:

- **Correctness:** `Newton` finds all isolated solutions to the system in the following sense: if $\langle v_1, \ldots, v_n \rangle$ is a solution, then `Newton` returns at least one box $\langle I_1, \ldots, I_n \rangle$ such that $v_i \in I_i \ (1 \leq i \leq n)$. In addition, `Newton` may guarantee the existence of a unique solution in some or all the boxes in the result. If the solutions are not isolated (e.g., the floating-point system is not precise enough to separate two solutions), then the boxes returned by the algorithm may contain several solutions.

- **Termination:** `Newton` always terminates.

- **Effectiveness:** `Newton` has been evaluated on a variety of benchmarks from kinematics, chemistry, combustion, economics, and mechanics. It outperforms the interval methods we are aware of and compares well with state-of-the-art continuation methods on many problems. Interestingly, `Newton` solves the Broyden banded function problem [8] and Moré-Cosnard discretization of a nonlinear integral equation [22] for several hundred variables.

- **Simplicity and Uniformity:** `Newton` is based on simple mathematical results and is easy to use and to implement. It is also based on a single concept: box-consistency.

The rest of this paper is organized as follows. Section 2 gives an overview of the approach. Section 3 contains the preliminaries. Section 4 presents an abstract version of the branch & prune algorithm. Section 5 discusses the implementation of the box-consistency. Section 6 describes the experimental results. Section 7 discusses related work and the development of the ideas presented here. Section 8 concludes the paper. A short version of this paper is available in [33].

2 Overview of The Approach

As mentioned, `Newton` is a global search algorithm which solves a problem by dividing it into subproblems which are solved recursively. In addition, `Newton` is a branch & prune algorithm which means that it is best viewed as an iteration of two steps

1. pruning the search space;

2. making a nondeterministic choice to generate two subproblems

until one or all solutions to a given problem are found.

The pruning step ensures that some local consistency conditions are satisfied. It consists of reducing the intervals associated with the variables so that every constraint appears to be locally consistent. The local consistency condition of `Newton` is called box-consistency, an approximation of arc-consistency, a notion well-known in artificial intelligence [16, 18] and used in many systems (e.g., [32, 34, 30]) to solve discrete combinatorial search problems. Informally speaking, a constraint is arc-consistent if for each value in the range of a variable there exist values in the ranges of the other variables such that the constraint is satisfied. `Newton` approximates arc-consistency which cannot be computed on real numbers in general.

The pruning step either fails, showing the absence of solution in the intervals, or succeeds in enforcing the local consistency condition. Sometimes, local consistency also implies global consistency as in the case of the Broyden banded function, i.e.,

$$f_i(x_1, \ldots, x_n) = x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j) \quad (1 \leq i \leq n)$$

where $J_i = \{j \mid j \neq i \;\&\; max(1, i-5) \leq j \leq min(n, i+1)\}$. The pruning step of `Newton` solves this problem in essentially linear time for initial intervals of the form $[-10^8, 10^8]$ and always proves the existence of a solution in the final box. However, in general, local consistency does not imply global consistency either because there are multiple solutions or simply because the local consistency condition is too weak. Consider the intersection of a circle and of a parabola:

$$\begin{cases} x_1^2 + x_2^2 = 1 \\ x_1^2 - x_2 = 0 \end{cases}$$

with initial intervals in $[-10^8, 10^8]$ and assume that we want the resulting intervals to be of size 10^{-8} or smaller. The pruning step returns the intervals

$$x_1 \in [-1.0000000000012430057, +1.0000000000012430057]$$
$$x_2 \in [-0.0000000000000000000, +1.0000000000012430057]$$

Informally speaking, `Newton` obtains the above pruning as follows. The first constraint is used to reduce the interval of x_1 by searching for the leftmost and rightmost "zeros" of the interval function

$$X_1^2 + [-10^8, 10^8]^2 = 1$$

These zeros are -1 and 1 and hence the new interval for x_1 becomes [-1,1] (modulo the numerical precision).[1] The same reasoning applies to x_2. The second constraint can be used to reduce further the interval of x_2 by searching for the leftmost and rightmost zeros of

$$[-1, 1]^2 - X_2 = 0$$

producing the interval [0,1] for x_2. No more reduction is obtained by `Newton` and branching is needed to make progress. Branching on x_1 produces the intervals

[1] The precision of the pruning step depends both on the numerical precision and the improvement factor selected (see Section 5.6) which explains the digits 12430057 in the above results.

$x_1 \in [-1.0000000000012430057, +0.0000000000000000000]$
$x_2 \in [-0.0000000000000000000, +1.0000000000012430057]$

Further pruning is obtained by taking the Taylor extension of the polynomials in the above equations around $(-0.5, 0.5)$, the center of the above box and by conditioning the system. `Newton` then uses these new constraints to find their leftmost and rightmost zeros as was done previously to obtain.

$x_1 \in [-1.0000000000000000000, -0.6049804687499998889]$
$x_2 \in [+0.3821067810058591529, +0.8433985806150308129].$

Additional pruning using the original statement of the constraints leads to the first solution

$x_1 \in [-0.7861513777574234974, -0.7861513777574231642]$
$x_2 \in [+0.6180339887498946804, +0.6180339887498950136].$

Backtracking on the choice for x_1 produces the intervals

$x_1 \in [-0.0000000000000000000, +1.0000000000012430057]$
$x_2 \in [-0.0000000000000000000, +1.0000000000012430057]$

and to the second solution

$x_1 \in [+0.7861513777574231642, +0.7861513777574233864]$
$x_2 \in [+0.6180339887498946804, +0.6180339887498950136].$

Note that, in this case, `Newton` makes the smallest number of choices to isolate the solutions.[2] To conclude this motivating section, let us illustrate `Newton` on a larger example which describes the inverse kinematics of an elbow manipulator [11]:

$$\begin{cases} s_2c_5s_6 - s_3c_5s_6 - s_4c_5s_6 + c_2c_6 + c_3c_6 + c_4c_6 = 0.4077 \\ c_1c_2s_5 + c_1c_3s_5 + c_1c_4s_5 + s_1c_5 = 1.9115 \\ s_2s_5 + s_3s_5 + s_4s_5 = 1.9791 \\ c_1c_2 + c_1c_3 + c_1c_4 + c_1c_2 + c_1c_3 + c_1c_2 = 4.0616 \\ s_1c_2 + s_1c_3 + s_1c_4 + s_1c_2 + s_1c_3 + s_1c_2 = 1.7172 \\ s_2 + s_3 + s_4 + s_2 + s_3 + s_2 = 3.9701 \\ s_i^2 + c_i^2 = 1 \qquad (1 \leq i \leq 6). \end{cases}$$

and assumes that the initial intervals are in $[-10^8, 10^8]$ again. The pruning step returns the intervals

$[-1.0000000000000000000, +1.0000000000000000000]$
$[-1.0000000000000000000, +1.0000000000000000000]$
$[+0.3233666666666665800, +1.0000000000000000000]$
$[-1.0000000000000000000, +1.0000000000000000000]$
$[-0.0149500000000000189, +1.0000000000000000000]$
$[-1.0000000000000000000, +1.0000000000000000000]$
$[-0.0209000000000001407, +1.0000000000000000000]$

[2] This example can also be solved by replacing x_1^2 in the first equation by x_2 to obtain a univariate constraint in x_2 which can be solved independently. However, this cannot always be done and the discussion here is what `Newton` would do, without making such optimizations.

$[-1.0000000000000000000, +1.0000000000000000000]$
$[+0.6596999999999998420, +1.0000000000000000000]$
$[-0.7515290480087772896, +0.7515290480087772896]$
$[-1.0000000000000000000, +1.0000000000000000000]$
$[-1.0000000000000000000, +1.0000000000000000000]$

showing already some interesting pruning. After exactly 12 branchings and in less than a second, `Newton` produces the first box with a proof of existence of a solution in the box.

3 Preliminaries

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, 8, 7, 19, 20]). Our definitions are slightly non-standard.

3.1 Interval Arithmetic

We consider $\Re^{\infty} = \Re \cup \{-\infty, \infty\}$ the set of real numbers extended with the two infinity symbols and the natural 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

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

The set of intervals is denoted by $\mathcal{I}$ and is ordered by set inclusion.[3]

Definition 2 [Enclosure and Hull] 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 $r \in \Re$. The interval hull of I_1 and I_2, denoted by $I_1 \uplus I_2$, is defined as $box\{I_1 \cup I_2\}$.

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 $\mathcal{I} \to \mathcal{I}$) 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. Finally, we use the following notations.

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

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

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

Definition 3 [Interval Extension] $F : \mathcal{I}^n \to \mathcal{I}$ is an interval extension of $f : \Re^n \to \Re$ iff

$$\forall I_1 \ldots I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow f(r_1, \ldots, r_n) \in F(I_1, \ldots, I_n).$$

An interval relation $C : \mathcal{I}^n \to \mathit{Bool}$ is an interval extension of a relation $c : \Re^n \to \mathit{Bool}$ iff

$$\forall I_1 \ldots I_n \in \mathcal{I} : [\ \exists r_1 \in I_1, \ldots, \exists r_n \in I_n\ c(r_1, \ldots, r_n)\ ] \Rightarrow C(I_1, \ldots, I_n).$$

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 $\doteq$ defined as

$$I_1 \doteq I_2 \ \Leftrightarrow\ (I_1 \cap I_2 \neq \emptyset)$$

is an interval extension of the equality relation on real numbers.

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 interval extensions for the basic real operators $+, -, \times$ and exponentiation (for instance, the interval extension of $+$ is defined by $\oplus$) and the basic real relations $=, \geq$. 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.

3.2 Division and Unions of Intervals

Even though many basic real operators can be naturally extended to work on intervals, division creates problems if the denominator interval includes 0. A tight extension of division to intervals can be best expressed if its result is allowed to be a union of intervals [10, 6, 12]. Assuming that $c \leq 0 \leq d$ and $c < d$, $[a, b]/[c, d]$ is defined as follows:

$$\begin{array}{ll}
[\lfloor b/c \rfloor, \infty] & \text{if } b \leq 0 \text{ and } d = 0 \\
[-\infty, \lceil b/d \rceil] \cup [\lfloor b/c \rfloor, \infty] & \text{if } b \leq 0 \text{ and } c < 0 < d \\
[-\infty, \lceil b/d \rceil] & \text{if } b \leq 0 \text{ and } c = 0 \\
[-\infty, \infty] & \text{if } a < 0 < b \\
[-\infty, \lceil a/c \rceil] & \text{if } a \geq 0 \text{ and } d = 0 \\
[-\infty, \lceil a/c \rceil] \cup [\lfloor b/c \rfloor, \infty] & \text{if } a \geq 0 \text{ and } c < 0 < d \\
[\lfloor a/d \rfloor, \infty] & \text{if } a \geq 0 \text{ and } c = 0.
\end{array}$$

When $[c, d] = [0, 0]$, $[a, b]/[c, d] = [-\infty, \infty]$. The case where $0 \notin [c, d]$ is easy to define. Note also that other operations such as addition and subtraction can also be extended to work with unions of intervals.

A typical use of unions of intervals in interval arithmetic is the intersection of an interval I with the result of an operation of the form $I_c - I_n/I_d$ to produce a new interval I'. To increase precision, $I_c - I_n/I_d$ is computed with unions of intervals producing a result of the form I_1 or $I_1 \cup I_2$. This result should be intersected with I to produce a single interval as result (i.e., not a union of intervals). A generalized intersection operation $\sqcap$ defined as

$$(I_1 \cup \ldots \cup I_n) \sqcap (I'_1 \cup \ldots \cup I'_m) = (I_1 \cap I'_1) \uplus (I_1 \cap I'_2) \uplus \ldots \uplus (I_n \cap I'_m)$$

is used for this purpose, i.e., we compute

$$I \sqcap (I_c - I_n/I_d).$$

This result may be more precise than

$$(I_1 \uplus \ldots \uplus I_n) \cap (I'_1 \uplus \ldots \uplus I'_m).$$

To illustrate the gain of precision, consider $I = [-5, -1]$, $I_1 = [-\infty, -3]$ and $I_2 = [3, \infty]$. The expression $I \sqcap (I_1 \cup I_2)$ evaluates to the interval $[-5, -3]$, while the expression $I \cap (I_1 \uplus I_2)$ returns the interval $[-5, -1]$, since $I_1 \uplus I_2 = [-\infty, \infty]$.

3.3 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 formal language by composing real variables, rational numbers, some predefined real numbers (e.g., π), arithmetic operations such as $+, -, \times$ and exponentiation to a natural number, parentheses, and relation symbols such as $\leq, =$.[4] We will abuse notation by denoting functions (resp. constraints) and their representations by the same symbol. Real variables in constraints will be taken from a finite (but arbitrary large) set $\{x_1, \ldots, x_n\}$. Similar conventions apply to interval functions and constraints. Interval variables will be taken from a finite (but arbitrary large) set $\{X_1, \ldots, X_n\}$, interval functions will be denoted by the letter F and interval constraints by the letter C. For simplicity of exposition, we restrict attention to equations. It is straightforward to generalize our results to inequalities (see Section 5.6).

4 The Branch & Prune Algorithm

This section describes the branch & prune algorithm `Newton`. Section 4.1 defines box-consistency, the key concept behind our algorithm. Section 4.2 shows how box-consistency can be instantiated to produce various pruning operators achieving various tradeoffs between accuracy and efficiency. Section 4.3 defines a conditioning operator used in `Newton` to improve the effectiveness of box-consistency. Section 4.4 specifies the pruning in `Newton`. Section 4.5 describes the algorithm. Recall that we assume that all constraints are defined over variables $x_1, \ldots, x_n$.

[4] It is easy to extend the language to include functions such as *sin, cos, e,* ....

4.1 Box Consistency

Box-consistency [2] is an approximation of arc-consistency, a notion well-known in artificial intelligence [16] 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 4 [Projection Constraint] A projection constraint $\langle c, i \rangle$ is a pair of a constraint c and an index i $(1 \leq i \leq n)$. Projection constraints are denoted by the letter p, possibly subscripted.

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 5 [Arc-Consistency] A projection constraint $\langle c, i \rangle$ is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i = D_i \cap \{r_i \mid \exists r_1 \in D_1, \ldots, \exists r_{i-1} \in D_{i-1}, \exists r_{i+1} \in D_{i+1}, \ldots, \exists r_n \in D_n : c(r_1, \ldots, r_n)\}$. 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 very effective on discrete combinatorial problems. However, it cannot be computed in general when working with real numbers and polynomial constraints. Moreover, simple approximations to take 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.,

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

This condition, used in systems like [26, 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 unpractical 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 6 [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 7 [Box-Consistency] An interval projection constraint $\langle C, i\rangle$ is box-consistent wrt $\vec{I} = \langle I_1, \ldots, I_n\rangle$ iff

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

where $l = \mathit{left}(I_i)$ and $u = \mathit{right}(I_i)$ An interval constraint is box-consistent wrt $\vec{I}$ if each of its projections is box-consistent wrt $\vec{I}$. A system of interval constraints is box-consistent wrt $\vec{I}$ iff each interval constraint in the system is box-consistent wrt $\vec{I}$.

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. Note also that the above condition is equivalent to

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

which shows clearly that box-consistency is an approximation of arc-consistency.[5] 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 wrt $\langle[-1, 1], [-1, 1]\rangle$ 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 $X_1 + X_2 - X_1 = 0$ is box-consistent, since $([-1, 1] + [-1, -1^+] - [-1, 1]) \cap [0, 0]$ and $([-1, 1] + [1^-, 1] - [-1, 1]) \cap [0, 0]$ are non-empty.

4.2 Interval Extensions for Box Consistency

Box-consistency strongly depends on the interval extensions chosen for the constraints and different interval extensions can produce very different (often incomparable) tradeoffs between pruning and computational complexity. In this section, we consider three extensions used in `Newton`: the natural interval extension, the distributed interval extension, and the Taylor interval extension.

4.2.1 Natural Interval Extension

The simplest extension of a function (resp. of a 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 by $\widehat{f}$ (resp. $\widehat{c}$) its natural extension.

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 $X_1(X_2 + X_3) \dot{=} \overline{0}$.

[5] It is interesting to note that this definition is also related to the theorem of Miranda [25]. In this case, box-consistency can be seen as replacing universally quantified variables by the intervals on which they range.

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. A very nice application where this extension is fundamental is the Moré-Cosnard discretization of a nonlinear integral equation (See Section 6.2). Using the natural extension allows users to minimize the problem of dependency of interval arithmetic and hence to increase precision.

4.2.2 Distributed Interval Extension

The second interval extension used by `Newton` does not preserve the way constraints are written but uses a distributed form of the constraints. The key advantage of this extension is that it allows the algorithm to enforce box-consistency by applying the interval Newton method on univariate real functions. The real functions are derived from univariate interval constraints obtained by replacing all but one variable by their intervals. As a consequence, applying box-consistency will be particularly efficient, although the pruning may be weaker than for the natural extension due to the dependency problem of interval arithmetics.[6] Intuitively, the distributed interval extension should be viewed as a way to speed up the computation of box-consistency on the natural extension. However, it may happen that it gives more precision than the natural extension if users are not careful in stating their constraints.

Definition 8 [Distributed Form] A constraint c in (simplified) sum of products form

$$m_1 + \cdots + m_k = 0,$$

where each monomial m_i is of the form $qx_1^{e_1} \cdots x_n^{e_n}$ with $q \in \mathcal{Q}$ and $e_i \in \mathcal{N}$, is said to be in distributed form.[7]

Definition 9 [Distributed Interval Extension] The distributed interval extension of a function f (resp. constraint c) is the natural extension of its distributed form. The distributed interval extension of a function f (resp. of a constraint c) is denoted by $\widetilde{f}$ (resp. $\widetilde{c}$).

Example 6 [Distributed Interval Extension] The distributed interval extension of the function $x_1(x_2 + x_3)$ is the interval function $X_1X_2 + X_1X_3$. The distributed interval extension of the constraint $x_1(x_2 + x_3) = 0$ is the interval constraint $X_1X_2 + X_1X_3 = \overline{0}$.

4.2.3 Taylor Interval Extension

The last interval extension we introduce is based on the Taylor expansion around a point. This extension is an example of centered forms which are interval extensions introduced by Moore [19] and studied by many authors, since they have important properties. The Taylor 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.

[6] Note that it is not always necessary to go through the distributed form to obtain the above property but `Newton` adopts it for simplicity.

[7] The distributed version can easily be turned into a canonical representation for constraints.

Definition 10 [Taylor Interval Extension] Let c be a constraint $f = 0$, f be a function with continuous partial derivatives, $\vec{I}$ be a box $(I_1, \ldots, I_n)$, and m_i be the center of I_i. The Taylor interval extension of c wrt $\vec{I}$, denoted by $c^{t(\vec{I})}$, is the interval constraint

$$\widehat{f}(\overline{m_1}, \ldots, \overline{m_n}) + \sum_{i=1}^{n} \widehat{\frac{\partial f}{\partial x_i}}(I_1, \ldots, I_n)\ (X_i - \overline{m_i}) \quad = \quad \overline{0}.$$

In the current version of our system, the partial derivatives are computed numerically using automatic differentiation [27].

4.3 Conditioning

It is interesting to note that box-consistency on the Taylor interval extension is closely related to the Hansen-Segupta's operator [8], which is an improvement over Krawczyk's operator [15]. Hansen and Smith [9] also argued that these operators are more effective for a system $\{f_1 = 0, \ldots, f_n = 0\}$ wrt a box $\langle I_1, \ldots, I_n \rangle$ when the interval Jacobian

$$M_{ij} = \widehat{\frac{\partial f_i}{\partial x_j}}(I_1, \ldots, I_n)\ (1 \leq i, j \leq n)$$

is diagonally dominant, i.e.,

$$mig(M_{i,i}) \geq \sum_{j=1, j \neq i}^{n} mag(M_{i,j})$$

where

$$mig([l, u]) = min(\mid l \mid, \mid u \mid) \text{ and } mag([l, u]) = max(\mid l \mid, \mid u \mid).$$

They also suggest a conditioning which consists in multiplying the linear relaxation by a real matrix which is the inverse of the matrix obtained by taking the center of $M_{i,j}$. The resulting system is generally solved through Gauss-Seidel iterations, giving the Hansen-Segupta's operator. See also [13, 14] for an extensive coverage of conditioners. `Newton` exploits this idea to improve the effectiveness of box-consistency on the Taylor interval extension. The conditioning of `Newton` is abstracted by the following definition.

Definition 11 [Conditioning] Let $S = \{f_1 = 0, \ldots, f_n = 0\}$. A conditioning of S is a system $S' = \{f'_1 = 0, \ldots, f'_n = 0\}$ where

$$f'_i = \sum_{k=1}^{n} A_{ik} f_k$$

where $A_{ik} \in \mathcal{Q}$.

In its present implementation, `Newton` uses a conditioning $cond(\{f_1 = 0, \ldots, f_n = 0\}, \vec{I})$ which returns a system $\{f'_1 = 0, \ldots, f'_n = 0\}$ such that

$$f'_i = \sum_{k=1}^{n} A_{ik} f_k$$

where

$$
\begin{array}{rcl}
M_{ij} & = & \widehat{\frac{\partial f_i}{\partial x_j}}(I_1, \ldots, I_n) \ (1 \leq i, j \leq n) \\
B_{ij} & = & center(M_{ij}) \\
A & = & \begin{cases} B^{-1} & \text{if } B \text{ is not singular} \\ I & \text{otherwise.} \end{cases}
\end{array}
$$

Note that the computation of the inverse of B is obtained by standard floating-point algorithms and hence it is only an approximation of the actual inverse.

4.4 Pruning in `Newton`

We now describe the pruning of `Newton`. The key idea behind `Newton` is to apply box-consistency at each node of the search tree, i.e., `Newton` reduces the current intervals for the variables in such a way that the constraint system is box-consistent wrt the reduced intervals and no solution is removed. The pruning is performed by using narrowing operators deduced from the definition of box-consistency. These operators are used to reduce the interval of a variable using a projection constraint.

Definition 12 [Box-Narrowing] Let $\langle C, i\rangle$ be an interval projection constraint and $\langle I_1, \ldots, I_n\rangle$ be a box. The narrowing operator `BOX-NARROW` is defined as

$\texttt{BOX-NARROW}(\langle C, i\rangle, \langle I_1, \ldots, I_n\rangle) = \langle I_1, \ldots, I_{i-1}, I, I_{i+1}, \ldots, I_n\rangle$

where I is defined as the largest set included in I_i such that $\langle C, i\rangle$ is box-consistent with respect to $\langle I_1, \ldots, I_{i-1}, I, I_{i+1}, \ldots, I_n\rangle$.

Proposition 1 [Soundness of the Box-Narrowing] Let C be an interval extension of c, $\langle C, i\rangle$ be an interval projection constraint, $\langle I_1, \ldots, I_n\rangle$ be a box, and

$$\langle I_1, \ldots, I_{i-1}, I, I_{i+1}, \ldots, I_n\rangle = \texttt{BOX-NARROW}(\langle C, i\rangle, \langle I_1, \ldots, I_n\rangle).$$

Then

$$r_1 \in I_1, \ldots, r_n \in I_n \ \& \ c(r_1, \ldots, r_n) \ \Rightarrow \ r_i \in I$$

Proof Assume that $r_1 \in I_1, \ldots, r_n \in I_n$ and that $r_i \notin I$. Then either $r_i < l$ or $r_i > u$, where $I = [l, u]$. If $r_i < l$, we have that

$$C(I_1, \ldots, I_{i-1}, \overline{r_i}, I_{i+1}, \ldots, I_n)$$

by definition of an interval extension and

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

by hypothesis. Hence, C is box-consistent wrt $\langle I_1, \ldots, I_{i-1}, I \uplus \overline{r_i}, I_{i+1}, \ldots, I_n\rangle$, which contradicts our hypothesis that I is defined as the largest set included in I_i such that $\langle C, i\rangle$ is box-consistent with respect to $\langle I_1, \ldots, I_{i-1}, I, I_{i+1}, \ldots, I_n\rangle$. The case $r_i > u$ is similar. $\Box$

We are now in a position to define the pruning algorithm of `Newton` which consists essentially in applying the narrowing operators of each projection until no further reduction occurs. The

procedure PRUNE(**in** $\mathcal{S}$: Set of Constraint; **inout** $\vec{I}:\mathcal{I}^n$)
begin
  repeat
    $\vec{I}_p$:= $\vec{I}$;
    BOX-PRUNE($\{\langle\widehat{c},i\rangle \mid c\in\mathcal{S} \;\&\; 1\le i\le n\} \;\cup\; \{\langle\widetilde{c},i\rangle \mid c\in\mathcal{S} \;\&\; 1\le i\le n\}$,$\vec{I}$);
    BOX-PRUNE($\{\langle c^{t(\vec{I})},i\rangle \mid c\in cond(\mathcal{S},\vec{I}) \;\&\; 1\le i\le n\}$,$\vec{I}$);
  until $\vec{I}=\vec{I}_p$;
end

procedure BOX-PRUNE(**in** $\mathcal{P}$: Set of Interval Projection Constraint; **inout** $\vec{I}:\mathcal{I}^n$)
begin
  repeat
    $\vec{I}_p$:= $\vec{I}$;
    $\vec{I}$:= $\bigcap\{$BOX-NARROW$(P,\vec{I}) \mid P\in\mathcal{P}\}$
  until $\vec{I}=\vec{I}_p$;
end

Figure 1: Pruning in `Newton`

pruning algorithm is depicted in Figure 1. It first applies box-consistency on the natural and distributed extensions until no further reduction occurs and then applies box-consistency on the Taylor extension. The two steps are iterated until a fixpoint is reached. Termination of the algorithm is guaranteed since the set $\mathcal{F}$ is finite and thus the intervals can only be reduced finitely often.

4.5 The Branch and Prune Algorithm `Newton`

Figure 2 is a very high level description of the branch and prune algorithm, highlighting the control flow. The algorithm applies operation `PRUNE` on the initial box. If the resulting box is empty (which means that one of its components is empty), then there is no solution by Proposition 1. 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). Variables for splitting are chosen by `BRANCH` using a round-robin heuristic: if $\{x_1,\ldots,x_n\}$ is the set of variables, then the algorithm splits the variables in the order $x_1,x_2,\ldots,x_n$ and reiterates the process until a solution is found.

5 Implementation of Box-Consistency

Figure 2 is a precise description of the branch and prune algorithm which leaves open the implementation of procedure `BOX-NARROW`. The purpose of this section is to describe how this procedure is implemented in `Newton`. The basic idea of the implementation is to use a different implementation of procedure `BOX-NARROW` for each interval extension in order to exploit their specific properties. We

```
function BranchAndPrune(S: Set of Constraint; I0 : I^n): Set of I^n;
begin
   I := PRUNE(S,I0);
   if ¬ IsEmpty(I) then
      if IsSmallEnough(I) then
         return {I}
      else
         ⟨I1, I2⟩ := BRANCH(I);
         return BranchAndPrune(S,I1) ∪ BranchAndPrune(S,I2)
      endif
   else
      return ∅
   endif
end
```

Figure 2: The Branch and Prune Algorithm

thus present three procedures in the section: BOX-NARROW-NE, BOX-NARROW-DE, and BOX-NARROW-TE. In addition, it is more convenient to define them in terms of projection constraints (instead of in terms of interval projection constraints).[8] The rest of this section is organized as follows. We start by describing a basic tool used in the implementations, then describe the various narrowing operators, discuss how to prove the existence of solution, and conclude by some implementation issues.

5.1 Extreme Zeros of an Interval Function

Box-consistency can often be reduced to two subproblems which, informally speaking, consist in shrinking the left (resp. the right) of an interval I' to the leftmost (resp. rightmost) zero of a univariate interval function F in I'. The univariate interval function F is an extension of a real function f which is either univariate, in which case F is a traditional interval extension, or multivariate, in which case F is obtained by taking an interval extension of f and substituting all variables but one, say x_i, by their intervals. In addition, we will have at our disposal a function F', the "derivative" of F, which is either an interval extension of the derivative of f (univariate case) or an interval extension of the partial derivative of f wrt x_i in which all variables but x_i have been replaced by their intervals.

The subproblems can be computed using a variation of the univariate interval Newton method. The method uses the following property for pruning the search space

"every zero of f in I lies also in $N(F, F', I)$"

where

$$N(F, F', I) = I \ \sqcap \ [\overline{center(I)} - \frac{F(\overline{center(I)})}{F'(I)}].$$

[8]There is no difficult in modifying the algorithm of Figure 2 to accommodate this change.

```
function LNAR(F, F' : 𝓘 → 𝓘, I : 𝓘):  𝓘;
begin
    r := right(I);
    if 0 ∉ F(I) then
        return ∅;
    I := N*(F, F', I);
    if 0 ∈ F([left(I), left(I)⁺]) then
        return I ⊎ r̄
    else
        ⟨I₁, I₂⟩ := SPLIT(I);
        if LNAR(F, F', I₁) ≠ ∅ then
            return LNAR(F, F', I₁) ⊎ r̄
        else
            return LNAR(F, F', I₂) ⊎ r̄
    endif
end;
```

Figure 3: The Left-Narrowing Function LNAR

More precisely, the algorithm uses $N^*(F, F', I) = \bigcap_{i=0}^{\infty} I_i$ where

$$\begin{array}{lcl} I_0 & = & I \\ I_{i+1} & = & N(F, F', I_i) \quad (0 \leq i) \end{array}$$

Figure 3 depicts a simple function LNAR to shrink to the leftmost zero (function RNAR is defined similarly). It make uses of a function SPLIT which, given an interval I, returns two intervals I_1 and I_2 such that $I = I_1 \cup I_2$ and $left(I_2) = right(I_1)$. The algorithm first applies the pruning procedure. It terminates if the left zero has been found or the interval cannot contain a zero. Otherwise, the interval is split into two subintervals. The leftmost interval is explored first. If it does not contain a zero, the rightmost interval is explored. It is worthwhile mentioning that the reduction on the right bound must not be used as part of the result, since the function would not meet its specification in this case. Function RNAR is responsible for finding the rightmost zero.

5.2 Box-Consistency on the Natural Interval Extension

We are now in a position to define the narrowing operator for the natural interval extension. The basic idea is very simple. For a constraint $\langle f = 0, i\rangle$, it consists in taking an interval extension of f and replacing all variables x_j but x_i by their interval I_j to obtain a univariate function. The leftmost and rightmost zeros are then computed to produce the result.

Definition 13 [Narrowing Operator for the Natural Interval Extension] The narrowing operator for the natural interval extension is defined as follows:

BOX-NARROW-NE($\langle f = 0, i\rangle, \langle I_1, \ldots, I_n\rangle$) = $\langle I_1, \ldots, I_{i-1}, I, I_{i+1}, \ldots, I_n\rangle$
where
    I = LNAR(F, F', RNAR(F, F', I_i))

$$
\begin{aligned}
F(X) &= \widehat{f}(I_1, \ldots, I_{i-1}, X, I_{i+1}, \ldots, I_n) \\
F'(X) &= \widehat{\frac{\partial f}{\partial x_i}}(I_1, \ldots, I_{i-1}, X, I_{i+1}, \ldots, I_n)
\end{aligned}
$$

Example 7 Let c be the constraint $x_1^2 + x_2^2 - 1 = 0$ and $\vec{I}$ be $\langle[-1,1],[-1,1]\rangle$. The function F and F' for $i = 1$ in the above definition are defined as follows:

$$
\begin{aligned}
&F(X) = X^2 + [-1,1]^2 - 1. \\
&F'(X) = 2X.
\end{aligned}
$$

Box-consistency on the natural extension (and on the distributed extension as well) can be applied even if the function is not differentiable. It suffices to omit the application of operator N^* in the functions `LNAR` and `RNAR`.

5.3 Box-Consistency on the Distributed Interval Extension

We now turn to the narrowing operator for the distributed interval extension. Box-consistency on the distributed interval extension can be enforced by using the univariate interval function

$$F(X) = \widetilde{f}(I_1, \ldots, I_{i-1}, X, I_{i+1}, \ldots, I_n)$$

and by searching its extreme zeros. However, since the function is in distributed form, it is possible to do better by using an idea from [11]. The key insight is to sandwich F exactly between two univariate real functions f_l and f_u defined as follows:

$$
\begin{aligned}
f_l(x) &= \mathit{left}(F(\overline{x})) \\
f_u(x) &= \mathit{right}(F(\overline{x})).
\end{aligned}
$$

Note that box-consistency can now be enforced by searching the leftmost and rightmost zeros of some interval extensions, say F_l and F_u, of f_l and f_u using interval extensions, say F_l' and F_u', of their derivatives f_l' and f_u'. Of course, `LNAR` and `RNAR` can be used to find these extreme zeros.

The key advantage of the distributed interval extension is that it is easy to define the function f_l and f_u constructively. Let F be of the form

$$F(X) = I_1 X^{n_1} + \ldots + I_p X^{n_p}.$$

Function f_l is defined as

$$f_l(x) = \mathit{low}(I_1, x, n_1) + \ldots + \mathit{low}(I_p, x, n_p)$$

where

$$
\mathit{low}(I, x, n) = \begin{cases} \mathit{left}(I)\ x^n & \text{if } x \geq 0 \vee n \text{ is even} \\ \mathit{right}(I)\ x^n & \text{otherwise} \end{cases}
$$

Function f_u is defined as

$$f_u(x) = \mathit{high}(I_1, x, n_1) + \ldots + \mathit{high}(I_p, x, n_p)$$

where

$$
\mathit{high}(I, x, n) = \begin{cases} \mathit{right}(I)\ x^n & \text{if } x \geq 0 \vee n \text{ is even} \\ \mathit{left}(I)\ x^n & \text{otherwise} \end{cases}
$$

Example 8 Consider the function $x_1(x_1 + x_2) - 4$ and assume that x_1 and x_2 range over $[0, 1]$. The distributed interval extension is

$$X_1^2 + X_1 X_2 - 4.$$

The function F obtained by projecting the distributed interval extension on variable X_1 is

$$F(X) = X^2 + [0, 1]X - 4.$$

The corresponding functions f_l and f_u are

$$\begin{aligned} f_u(x) &= x^2 - 4 \\ f_l(x) &= x^2 + x - 4. \end{aligned}$$

Their natural interval extensions are, of course,

$$\begin{aligned} F_u(X) &= X^2 - 4 \\ F_l(X) &= X^2 + X - 4. \end{aligned}$$

The narrowing operator can now be obtained by using interval Newton method on the above two functions. Some care must be applied obviously since f_l and f_u are not differentiable at 0. The method is more efficient than applying the interval Newton method on the interval function, since intervals have been replaced by numbers, increasing the precision of the Newton operator N^*. We now present the narrowing operator.

Definition 14 [Narrowing Operator for the Distributed Interval Extension] Let $\langle f = 0, i\rangle$ be a projection constraint, let F, f_l, and f_u be the functions

$$\begin{array}{lcl} F(X) & = & \widetilde{f}(I_1, \ldots, I_{i-1}, X, I_{i+1}, \ldots, I_n) \\ f_l(x) & = & \mathit{left}(F(\overline{x})) \\ f_u(x) & = & \mathit{right}(F(\overline{x})) \\ F_l & = & \widehat{f_l} \\ F_u & = & \widehat{f_u} \\ F_l' & = & \widehat{f_l'} \\ F_u' & = & \widehat{f_u'} \\ l & = & \mathit{left}(I_i) \\ u & = & \mathit{right}(I_i) \end{array}$$

Assuming that $\langle I_1, \ldots, I_n\rangle$ is not empty, the narrowing operator for the distributed interval extension is defined as follows:

$\texttt{BOX-NARROW-DE}(\langle f = 0, i\rangle, \langle I_1, \ldots, I_n\rangle)$ = $\langle I_1, \ldots, I_{i-1}, [l_i, u_i], I_{i+1}, \ldots, I_n\rangle$
where

$$l_i = \begin{cases} l & \texttt{if } 0 \in F([l, l^+]) \\ \mathit{left}(\texttt{LNAR}(F_l, F_l', [l, 0]) \cup \texttt{LNAR}(F_l, F_l', [0, u])) & \texttt{if } F([l, l^+]) > 0 \\ \mathit{left}(\texttt{LNAR}(F_u, F_u', [l, 0]) \cup \texttt{LNAR}(F_u, F_u', [0, u])) & \texttt{otherwise} \end{cases}$$

$$u_i = \begin{cases} u & \texttt{if } 0 \in F([u^-, u]) \\ \mathit{right}(\texttt{RNAR}(F_l, F_l', [l, 0]) \cup \texttt{RNAR}(F_l, F_l', [0, u])) & \texttt{if } F([l, l^+]) > 0 \\ \mathit{right}(\texttt{RNAR}(F_u, F_u', [l, 0]) \cup \texttt{RNAR}(F_u, F_u', [0, u])) & \texttt{otherwise.} \end{cases}$$

5.4 Taylor Interval Extension

We conclude by presenting the narrowing operator for the Taylor interval extension. Box-consistency on the Taylor interval extension can be enforced by using the interval function

$$F(X) = f^{t(\langle I_1,\ldots,I_n\rangle)}(I_1,\ldots,I_{i-1},X,I_{i+1},\ldots,I_n)$$

and by applying a simple implementation of LNAR and RNAR where the Newton operator N^* is omitted. However, it is possible to do better by noticing that the constraint is of the form

$$\widehat{f}(\overline{m_1},\ldots,\overline{m_n}) + \sum_{j=1}^{i-1}\widehat{\frac{\partial f}{\partial x_j}}(\vec{I})(I_j - \overline{m_j}) + \widehat{\frac{\partial f}{\partial x_i}}(I_1,\ldots,I_n)(X_i - \overline{m_i}) + \sum_{j=i+1}^{n}\widehat{\frac{\partial f}{\partial x_j}}(\vec{I})(I_j - \overline{m_j}) = \overline{0}$$

where $m_i = \mathit{center}(I_i)$ and contains a single variable which can be isolated to compute box-consistency directly.

Definition 15 [Narrowing Operator for the Taylor Interval Extension] The narrowing operator for the Taylor interval extension is defined as follows:

$$\texttt{BOX-NARROW-TE}(\langle f=0,i\rangle,\langle I_1,\ldots,I_n\rangle) = \langle I_1,\ldots,I_{i-1},I,I_{i+1},\ldots,I_n\rangle$$

where

$$I = I_i \sqcap (\overline{m_i} - \frac{1}{\widehat{\frac{\partial f}{\partial x_i}}(I_1,\ldots,I_n)}[\sum_{j=1,j\neq i}^{n}\widehat{\frac{\partial f}{\partial x_j}}(I_1,\ldots,I_n)\ (I_j - \overline{m_j}) + \widehat{f}(\overline{m_1},\ldots,\overline{m_n})]).$$

and $m_i = \mathit{center}(I_i)$.

5.5 Existence of Solution

We now briefly describe how `Newton` proves the existence of solutions. No special effort has been devoted to this topic and the techniques could certainly be improved in various ways. Let $\{f_1 = 0,\ldots,f_n = 0\}$ be a conditioned system of equations over variables $\{x_1,\ldots,x_n\}$, let $\langle I_1,\ldots,I_n\rangle$ be a box and define the intervals I'_i $(1 \leq i \leq n)$ as follows:

$$I'_i = (\overline{m_i} - \frac{1}{\widehat{\frac{\partial f_i}{\partial x_i}}(I_1,\ldots,I_n)}[\sum_{j=1,j\neq i}^{n}\widehat{\frac{\partial f_i}{\partial x_j}}(I_1,\ldots,I_n)\ (I_j - \overline{m_j}) + \widehat{f}_i(\overline{m_1},\ldots,\overline{m_n})])$$

and $m_i = \mathit{center}(I_i)$. If

$$\langle I'_1,\ldots,I'_n\rangle \subseteq \langle I_1,\ldots,I_n\rangle$$

then there exists a unique zero in $\langle I'_1,\ldots,I'_n\rangle$. A proof of this result can be found in [25] where credit is given to Moore and Nickel. Note also the intervals I'_i have to be computed for box-consistency on the Taylor interval extension.

5.6 Implementation Issues

We now review some implementation issues which arise in programming the algorithm.

Priorities In the pruning algorithm, it is important for efficiency reasons to use a priority queue to ensure that projections over the distributed interval extension are selected before projections over the natural interval extension. `Newton` also does not enforce box-consistency on the distributed version whenever it is believed to lose too much precision (e.g., an expression raised to some power).

Precision In practice, it is often sufficient to return intervals whose widths[9] are within the desired accuracy instead of returning intervals of the form $[l, l^+]$. It is easy to modify the `BRANCH` operation to split only intervals whose widths is above the required accuracy. Our system allows users to specify the accuracy.

Improvement Factor Box-consistency can sometimes take much time to remove small parts of the intervals. In these cases, it is probably more cost-effective to branch. Once again, it is easy to modify the algorithm to avoid this problem by making sure that the narrowing operators do not update the intervals unless some significant reduction has taken place. Since the notion of significant reduction may be problem-dependent, our system lets users specify the improvement factor necessary to update an interval in a projection.

Automatic Differentiation As mentioned, our algorithm takes a very simple approach to obtain partial derivatives, i.e., no effort is spent in factoring common expressions to reduce the dependency problem of interval arithmetic. The main reason comes from the fact that we are using automatic differentiation [27] to evaluate the derivatives together with the functions. This choice may be reconsidered in a future version of the system.

Inequalities It is straightforward to generalize the above algorithms for inequalities. In general, it suffices to test if the inequality is satisfied at the end points of the interval. If it is not, then the problem reduces once again to finding the leftmost and/or rightmost zeros.

6 Experimental Results

This section reports experimental results of `Newton` on a variety of standard benchmarks. The benchmarks were taken from papers on numerical analysis [22], interval analysis [8, 11, 21], and continuation methods [35, 24, 23, 17]. We also compare `Newton` with a traditional interval method using the Hansen-Segupta's operator, range testing, and branching. This method uses the same implementation technology as `Newton` and is denoted by `HRB` in the following.[10] Finally, we compare `Newton` with a state-of-the-art continuation method [35], denoted by `CONT` in the following. Note that all results given in this section were obtained by running `Newton` on a `Sun Sparc 10` workstation to obtain all solutions. In addition, the final intervals must have widths smaller than 10^{-8} and `Newton` always uses an improvement factor of 10%. The results are summarized in Table 1. For each benchmark, we give the number of variables (n), the total degree of the system (d), the initial range for the variables, and the results of each method in seconds. Note that the times

[9] The width of $[l, u]$ is $u - l$.

[10] Some interval methods such as [7] are more sophisticated than `HRB` but the sophistication aims at speeding up the computation near a solution. Our main contribution is completely orthogonal and aims at speeding up the computation when far from a solution and hence comparing it to `HRB` is meaningful.

Benchmarks	v	d	range	`Newton`	`HRB`	`CONT`
Broyden	10	3^{10}	[-1,1]	1.65	18.23	
Broyden	20	3^{20}	[-1,1]	4.25	?	
Broyden	320	3^{320}	[-1,1]	113.71	?	
Broyden	320	3^{320}	$[-10^8, 10^8]$	143.40	?	
Moré-Cosnard	20	3^{20}	$[-4, 5]$	24.49	968.25	
Moré-Cosnard	40	3^{40}	$[-4, 5]$	192.81	?	
Moré-Cosnard	80	3^{80}	$[-4, 5]$	1752.64	?	
Moré-Cosnard	80	3^{80}	$[-10^8, 0]$	1735.09	?	
i1	10	3^{10}	[-2,2]	0.06	14.28	
i2	20	3^{20}	[-1,2]	0.30	1821.23	
i3	20	3^{20}	[-2,2]	0.31	5640.80	
i4	10	6^{10}	[-1,1]	73.94	445.28	
i5	10	11^{10}	[-1,1]	0.08	33.58	
kin1	12	4608	$[-10^8, 10^8]$	14.24	1630.08	
kin2	8	256	$[-10^8, 10^8]$	353.06	4730.34	35.61
eco	4	18	$[-10^8, 10^8]$	0.60	2.44	1.13
eco	5	54	$[-10^8, 10^8]$	3.35	29.88	5.87
eco	6	162	$[-10^8, 10^8]$	22.53	?	50.18
eco	7	486	$[-10^8, 10^8]$	127.65	?	991.45
eco	8	1458	$[-10^8, 10^8]$	915.24	?	
eco	9	4374	$[-10^8, 10^8]$	8600.28	?	
combustion	10	96	$[-10^8, 10^8]$	9.94	?	57.40
chemistry	5	108	$[0, 10^8]$	6.32	?	56.55
neuro	6	1024	$[-10, 10]$	0.91	28.84	5.02
neuro	6	1024	$[-1000, 1000]$	172.71	?	5.02

Table 1: Summary of the Experimental Results

for the continuation method are on a `DEC 5000/200`. A space in a column means that the result is not available for the method. A question mark means that the method does not terminate in a reasonable time (> 1 hour). The rest of the section describes each benchmark and the results in much more detail. For each benchmark, we report the CPU times in seconds, the growth of the CPU time, the number of branch operations *branching*, the number of narrowings on the various extensions *na-ne, na-ee, na-te*, the total number of narrowings *na-tot*, the number of function evaluations (including evaluation of derivatives which are counted as normal function evaluations) for each of the extensions *fe-ne, fe-ee, fe-te* and the total number of function evaluations *fe-tot*. We also indicate the number of preconditionings by *pr-con* and whether the algorithm can prove the existence of the solutions in the resulting intervals by *proof*.

6.1 Broyden Banded Functions

This is a traditional benchmark of interval techniques and was used for instance in [7]. It consists in finding the zeros of the functions

$$f_i(x_1, \ldots, x_n) = x_i(2 + 5x_i^2) + 1 - \textstyle\sum_{j \in J_i} x_j(1 + x_j) \quad (1 \le i \le n)$$

where $J_i = \{j \mid j \neq i \;\&\; max(1, i-5) \le j \le min(n, i+1)\}$. One of the interesting features of this benchmark is that it is easy to scale up to an arbitrary dimension and hence provides a good

	5	10	20	40	80	160	320
CPU time	0.20	1.65	4.25	9.79	22.13	48.30	113.71
growth		8.25	2.57	2.30	2.26	2.18	2.35
branching	0	0	0	0	0	0	0
na-ne	57	260	661	1607	4351	8096	17126
na-ee	1226	8334	21236	48540	102797	206926	414798
na-te	35	110	260	560	1200	2400	4480
na-tot	1318	8704	22157	50707	108348	217422	436404
fe-ne	81	1828	2943	5103	11993	20431	42125
fe-ee	3462	21518	53722	121984	257398	517640	1036210
fe-te	95	320	920	2720	8800	30400	111360
fe-tot	3638	23666	57585	129807	278191	568471	1189695
pr-con	0	0	0	0	0	0	0
proof	yes	yes	yes	yes	yes	yes	yes

Table 2: `Newton` on the Broyden Banded functions with initial intervals $[-1, 1]$

	5	10	20	40	80	160	320
CPU time	0.31	2.15	5.09	12.49	27.69	61.60	143.40
growth		6.93	2.36	2.45	2.21	2.22	2.32
branching	0	0	0	0	0	0	0
pr-con	0	0	0	0	0	0	0
proof	yes	yes	yes	yes	yes	yes	yes

Table 3: `Newton` on the Broyden Banded functions with initial intervals $[-10^8, 10^8]$

basis to compare various methods. Table 2 reports the results of our algorithm for various sizes assuming initial intervals $[-1, 1]$.

The results indicate that `Newton` solves the problem using only constraint propagation: no branching is needed. In addition, the growth of the computation times is very low and indicates that `Newton` is essentially linear and can thus solve very large instances of this problem. Finally, `Newton` proves the existence of a solution in the final intervals. To our knowledge, no other algorithm has all these functionalities. Table 3 shows the same results when the initial intervals are $[-10^8, 10^8]$. They indicate that the CPU time increases only slightly in this problem when the initial intervals become substantially larger. It is interesting to note that substantial pruning is obtained by box-consistency on the natural and distributed extensions alone. For $n = 10$, maximal box-consistency on these two extensions produces the intervals

$$
\begin{array}{l}
[-0.4283028737061274627, -0.4283028534683728794] \\
[-0.4765964317901201786, -0.4765964169224605195] \\
[-0.5196524683730758821, -0.5196524589206473754] \\
[-0.5580993358758108425, -0.5580993137885511545] \\
[-0.5925061654931400579, -0.5925061481657747375] \\
[-0.6245036923913307448, -0.6245036720076052594] \\
[-0.6232394806883442274, -0.6232394621928379896] \\
[-0.6213938520278742273, -0.6213938315652728361] \\
[-0.6204536054436834425, -0.6204535878744913413] \\
[-0.5864692773020701023, -0.5864692641387999616]
\end{array}
$$

	5	10	20	40	80
CPU time	0.75	4.07	24.49	192.81	1752.64
growth		5.42	6.02	7.87	9.08
branching	0	0	0	0	0
na-ne	3663	12616	46555	213949	1236532
na-te	104	255	505	1005	2807
na-tot	3767	12871	47060	214954	1239339
fe-ne	8775	31837	107977	466595	2586907
fe-te	884	3111	11211	42411	166415
fe-tot	9659	34948	119188	509006	2753322
fe-grow		3.62	3.41	4.27	5.40
pr-con	1	1	1	1	1
proof	yes	no	no	no	no

Table 4: `Newton` on the Moré-Cosnard nonlinear integral Equation with initial intervals in $[-4, 5]$

which have widths lower than 10^{-6}. Note that the Hansen-Segupta's operator alone does not produce any pruning initially and returns the initial intervals whether they be of the form $[-10^8, +10^8]$ or $[-1, 1]$. This indicates that box-consistency on the natural and distributed interval extensions are particularly effective when far from a solution while box-consistency on the Taylor extension (and the Hansen-Segupta's operator) is effective when near a solution.

It is also interesting to stress the importance of box-consistency on the natural extension in this example to reduce the growth factor. Without it, the algorithm takes about 48 and 440 seconds instead of 27 and 61 for `Newton` for $n = 80$ and $n = 160$, since the distributed interval extension loses precision due to the dependency problem.

Finally, it is interesting to compare `Newton` with traditional interval methods. `HRB` takes 0.34 seconds on $n = 5$ with 18 branchings, about 18 seconds for $n = 10$ with about 300 branchings, and does not return after more than an hour on $n = 20$.

6.2 Discretization of a Nonlinear Integral Equation

This example comes from [22] and is also a standard benchmark for nonlinear equation solving . It consists in finding the root of the functions $f_k(x_1, \ldots, x_m)$ $(1 \leq k \leq m)$ defined as

$$x_k + \frac{1}{2(m+1)}[(1-t_k)\sum_{j=1}^{k} t_j(x_j + t_j + 1)^3 + t_k \sum_{j=k+1}^{m} (1-t_j)(x_j + t_j + 1)^3]$$

where $t_j = jh$ and $h = 1/(m+1)$. These functions come from the discretization of a nonlinear integral equation, giving a constraint system denser than the sparse constraint system for the Broyden banded functions. The variables x_i were given initial domains $[-4, 5]$ as in [28] and the computation results are given in Table 4.

Once again, it is interesting to note that `Newton` is completely deterministic on this problem, i.e., it does not do any branching. `Newton` is probably cubic in the number of variables for this problem. It is important to point out the critical role of box-consistency on the natural extension to solve this problem efficiently. `Newton` without the natural extension would not be deterministic and would slow down exponentially, since box-consistency on the distributed extension loses too

	5	10	20	40	80
CPU time	0.70	3.82	20.81	189.94	1735.09
growth		5.45	5.44	9.12	9.13
branching	0	0	0	0	0
pr-con	0	0	0	0	0
proof	yes	no	no	no	no

Table 5: `Newton` on the Moré-Cosnard nonlinear integral Equation with initial intervals in $[-10^8, 0]$

	5	10	20	40	80
CPU time	0.66	7.76	968.25	?	?
branching	5	24	508	?	?
fe-tot	3709	20194	1285764	?	?
pr-con	7	32	667	?	?
proof	yes	no	no	?	?

Table 6: The `HRB` algorithm on the Moré-Cosnard nonlinear integral Equation with initial intervals in $[-4, 5]$

much precision due to the dependency problem (multiple occurrences of the same variable) and box-consistency on the Taylor interval extension is not helpful initially. Once again, we observe that box-consistency over the natural extension is helpful when far from a solution while box-consistency on the Taylor extension is useful to terminate the search quickly. Table 5 gives the result for the initial intervals of size $[-10^8, 0]$, which shows that the algorithm continues to perform well in this case. Finally, Table 6 gives the results for the `HRB` algorithm on this problem. Once again, `Newton` outperforms the `HRB` method substantially.

6.3 Interval Arithmetic Benchmarks

This section considers standard benchmarks from interval arithmetic papers [20, 11]. Benchmark `i1` is the following set of equations

$$\left\{\begin{array}{l}
0 = x_1 - 0.25428722 - 0.18324757\ x_4\ x_3\ x_9 \\
0 = x_2 - 0.37842197 - 0.16275449\ x_1\ x_{10}\ x_6 \\
0 = x_3 - 0.27162577 - 0.16955071\ x_1\ x_2\ x_{10} \\
0 = x_4 - 0.19807914 - 0.15585316\ x_7\ x_1\ x_6 \\
0 = x_5 - 0.44166728 - 0.19950920\ x_7\ x_6\ x_3 \\
0 = x_6 - 0.14654113 - 0.18922793\ x_8\ x_5\ x_{10} \\
0 = x_7 - 0.42937161 - 0.21180486\ x_2\ x_5\ x_8 \\
0 = x_8 - 0.07056438 - 0.17081208\ x_1\ x_7\ x_6 \\
0 = x_9 - 0.34504906 - 0.19612740\ x_{10}\ x_6\ x_8 \\
0 = x_{10} - 0.42651102 - 0.21466544\ x_4\ x_8\ x_1
\end{array}\right.$$

with initial intervals $[-2, 2]$. Benchmark `i2` is the set of equations

$$
\left\{
\begin{array}{l}
0 = x_1 - 0.24863995 - 0.19594124\ x_7\ x_{10}\ x_{16} \\
0 = x_2 - 0.87528587 - 0.05612619\ x_{18}\ x_8\ x_{11} \\
0 = x_3 - 0.23939835 - 0.20177810\ x_{10}\ x_7\ x_{11} \\
0 = x_4 - 0.47620128 - 0.16497518\ x_{12}\ x_{15}\ x_1 \\
0 = x_5 - 0.24711044 - 0.20198178\ x_8\ x_9\ x_{16} \\
0 = x_6 - 0.33565227 - 0.15724045\ x_{16}\ x_{18}\ x_{11} \\
0 = x_7 - 0.13128974 - 0.12384342\ x_{12}\ x_{13}\ x_{15} \\
0 = x_8 - 0.45937304 - 0.18180253\ x_{19}\ x_{15}\ x_{18} \\
0 = x_9 - 0.46896600 - 0.21241045\ x_{13}\ x_2\ x_{17} \\
0 = x_{10} - 0.57596835 - 0.16522613\ x_{12}\ x_9\ x_{13} \\
0 = x_{11} - 0.56896263 - 0.17221383\ x_{16}\ x_{17}\ x_8 \\
0 = x_{12} - 0.70561396 - 0.23556251\ x_{14}\ x_{11}\ x_4 \\
0 = x_{13} - 0.59642512 - 0.24475135\ x_7\ x_{16}\ x_{20} \\
0 = x_{14} - 0.46588640 - 0.21790395\ x_{13}\ x_3\ x_{10} \\
0 = x_{15} - 0.10607114 - 0.20920602\ x_1\ x_9\ x_{10} \\
0 = x_{16} - 0.26516898 - 0.21037773\ x_4\ x_{19}\ x_9 \\
0 = x_{17} - 0.20436664 - 0.19838792\ x_{20}\ x_{10}\ x_{13} \\
0 = x_{18} - 0.56003141 - 0.18114505\ x_6\ x_{13}\ x_8 \\
0 = x_{19} - 0.92894617 - 0.04417537\ x_7\ x_{13}\ x_{16} \\
0 = x_{20} - 0.57001682 - 0.17949149\ x_1\ x_3\ x_{11}
\end{array}
\right.
$$

with initial intervals $[-1, 2]$. Benchmark `i3` has the same set of equations as `i2` but has initial intervals $[-2, 2]$. Benchmark `i4` has the set of equations

$$
\left\{
\begin{array}{l}
0 = x_1^2 - 0.25428722 - 0.18324757\ x_4^2\ x_3^2\ x_9^2 \\
0 = x_2^2 - 0.37842197 - 0.16275449\ x_1^2\ x_{10}^2\ x_6^2 \\
0 = x_3^2 - 0.27162577 - 0.16955071\ x_1^2\ x_2^2\ x_{10}^2 \\
0 = x_4^2 - 0.19807914 - 0.15585316\ x_7^2\ x_1^2\ x_6^2 \\
0 = x_5^2 - 0.44166728 - 0.19950920\ x_7^2\ x_6^2\ x_3^2 \\
0 = x_6^2 - 0.14654113 - 0.18922793\ x_8^2\ x_5^2\ x_{10}^2 \\
0 = x_7^2 - 0.42937161 - 0.21180486\ x_2^2\ x_5^2\ x_8^2 \\
0 = x_8^2 - 0.07056438 - 0.17081208\ x_1^2\ x_7^2\ x_6^2 \\
0 = x_9^2 - 0.34504906 - 0.19612740\ x_{10}^2\ x_6^2\ x_8^2 \\
0 = x_{10}^2 - 0.42651102 - 0.21466544\ x_4^2\ x_8^2\ x_1^2
\end{array}
\right.
$$

and initial intervals $[-1, 1]$. The number of solutions must be a multiple of 1024. Benchmark `i5` has the following set of equations

$$
\left\{
\begin{array}{l}
0 = x_1 - 0.25428722 - 0.18324757\ x_4^3\ x_3^3\ x_9^3\ +\ x_3^4\ x_9^7 \\
0 = x_2 - 0.37842197 - 0.16275449\ x_1^3\ x_{10}^3\ x_6^3\ +\ x_{10}^4\ x_6^7 \\
0 = x_3 - 0.27162577 - 0.16955071\ x_1^3\ x_2^3\ x_{10}^3\ +\ x_2^4\ x_{10}^7 \\
0 = x_4 - 0.19807914 - 0.15585316\ x_7^3\ x_1^3\ x_6^3\ +\ x_1^4\ x_6^7 \\
0 = x_5 - 0.44166728 - 0.19950920\ x_7^3\ x_6^3\ x_3^3\ +\ x_6^4\ x_3^7 \\
0 = x_6 - 0.14654113 - 0.18922793\ x_8^3\ x_5^3\ x_{10}^3\ +\ x_5^4\ x_{10}^7 \\
0 = x_7 - 0.42937161 - 0.21180486\ x_2^3\ x_5^3\ x_8^3\ +\ x_5^4\ x_8^7 \\
0 = x_8 - 0.07056438 - 0.17081208\ x_1^3\ x_7^3\ x_6^3\ +\ x_7^4\ x_6^7 \\
0 = x_9 - 0.34504906 - 0.19612740\ x_{10}^3\ x_6^3\ x_8^3\ +\ x_6^4\ x_8^7 \\
0 = x_{10} - 0.42651102 - 0.21466544\ x_4^3\ x_8^3\ x_1^3\ +\ x_8^4\ x_1^7
\end{array}
\right.
$$

	i1	i2	i3	i4	i5
CPU time	0.06	0.30	0.31	73.94	0.08
branching	0	0	0	1023	0
na-ee	625	2107	1949	290776	381
na-te	0	80	80	37930	21
na-tot	625	2187	2029	328706	401
fe-ee	1760	5698	5318	752220	992
fe-te	0	560	560	269760	140
fe-tot	1760	6258	5878	1021980	1132
pr-con	0	1	1	1939	1
proof	yes	yes	yes	yes	yes

Table 7: `Newton` on the Traditional Interval Arithmetic Benchmarks

	i1	i2	i3	i4	i5
CPU time	14.28	1821.23	5640.80	445.28	33.58
branching	498	9031	36933	11263	1173
fe-tot	77380	6441640	19979025	2554066	154948
pr-con	586	12817	42193	14335	1211
proof	yes	yes	yes	yes	yes

Table 8: `HRB` on the Traditional Interval Arithmetic Benchmarks

and initial intervals $[-1, 1]$.

`Newton` solves all the problems with one solution without branching and solves the problem having 1024 solutions with 1023 branchings. Note also that box-consistency on the distributed extension solves benchmark `i1` alone. The results once again confirm our observation on when the various extensions are useful. Closely related results were observed in [11] on these benchmarks (see the related work section for a more detailed comparison) but our algorithm is in general about 4 times faster (assuming similar machines) and does not do any branching on `i5`. Table 8 also describes the results for the traditional interval arithmetic method. The importance of box-consistency on the distributed extension can easily be seen from these results. Note also that `Newton` (and interval methods) can prove the existence of a solution in the final intervals for all these problems.

It is also interesting to note that problem `i4` can be solved dramatically more efficiently simply by introducing intermediary variables $y_i = x_i^2$. The excution times then dropped to less than 0.5 seconds.

6.4 Kinematics Applications

We now describe the performance of `Newton` on two kinematics examples. Application `kin1` comes from robotics and describes the inverse kinematics of an elbow manipulator [11]. It consists of a

- 0.249150680	+ 0.125016350	- 0.635550070	+ 1.48947730
+ 1.609135400	- 0.686607360	- 0.115719920	+ 0.23062341
+ 0.279423430	- 0.119228120	- 0.666404480	+ 1.32810730
+ 1.434801600	- 0.719940470	+ 0.110362110	- 0.25864503
+ 0.000000000	- 0.432419270	+ 0.290702030	+ 1.16517200
+ 0.400263840	+ 0.000000000	+ 1.258776700	- 0.26908494
- 0.800527680	+ 0.000000000	- 0.629388360	+ 0.53816987
+ 0.000000000	- 0.864838550	+ 0.581404060	+ 0.58258598
+ 0.074052388	- 0.037157270	+ 0.195946620	- 0.20816985
- 0.083050031	+ 0.035436896	- 1.228034200	+ 2.68683200
- 0.386159610	+ 0.085383482	+ 0.000000000	- 0.69910317
- 0.755266030	+ 0.000000000	- 0.079034221	+ 0.35744413
+ 0.504201680	- 0.039251967	+ 0.026387877	+ 1.24991170
- 1.091628700	+ 0.000000000	- 0.057131430	+ 1.46773600
+ 0.000000000	- 0.432419270	- 1.162808100	+ 1.16517200
+ 0.049207290	+ 0.000000000	+ 1.258776700	+ 1.07633970
+ 0.049207290	+ 0.013873010	+ 2.162575000	- 0.69686809

Table 9: Coefficients for the Inverse Kinematics Example.

sparse system with 12 variables and the set of equations is as follows:

$$\begin{cases} s_2c_5s_6 - s_3c_5s_6 - s_4c_5s_6 + c_2c_6 + c_3c_6 + c_4c_6 = 0.4077 \\ c_1c_2s_5 + c_1c_3s_5 + c_1c_4s_5 + s_1c_5 = 1.9115 \\ s_2s_5 + s_3s_5 + s_4s_5 = 1.9791 \\ c_1c_2 + c_1c_3 + c_1c_4 + c_1c_2 + c_1c_3 + c_1c_2 = 4.0616 \\ s_1c_2 + s_1c_3 + s_1c_4 + s_1c_2 + s_1c_3 + s_1c_2 = 1.7172 \\ s_2 + s_3 + s_4 + s_2 + s_3 + s_2 = 3.9701 \\ s_i^2 + c_i^2 = 1 \qquad (1 \leq i \leq 6). \end{cases}$$

The second benchmark, denoted by `kin2`, is from [23] and describes the inverse position problem for a six-revolute-joint problem in mechanics. The equations which describe a denser constraint system are as follows:

$$\begin{cases} x_i^2 + x_{i+1}^2 - 1 = 0 \qquad (1 \leq i \leq 4) \\ a_{1i}x_1x_3 + a_{2i}x_1x_4 + a_{3i}x_2x_3 + a_{4i}x_2x_4 + a_{5i}x_5x_7 + a_{6i}x_5x_8 + a_{7i}x_6x_7 + a_{8i}x_6x_8 \\ a_{9i}x_1 + a_{10i}x_2 + a_{11i}x_3 + a_{12i}x_4 + a_{13i}x_5a_{14i}x_6 + a_{15i}x_7 + a_{16i}x_8 + a_{17i} = 0 \;\; (1 \leq i \leq 4) \end{cases}$$

where the coefficients a_{ki} are given in table 9. In both examples, the initial intervals were given as $[-10^8, 10^8]$.

The results of `Newton` on these two benchmarks are given in Table 10. `Newton` is fast on the first benchmark and does not branch much to obtain all solutions. The algorithm in [11] branches more (the reported figure is 257 branches but it is not really comparable due to the nature of the algorithm) and is about 16 times slower on comparable machines. We are not aware of the results of continuation methods on this problem. `Newton` is slower on the second application and takes about 6 minutes. The continuation method described in [35] requires about 30 seconds on a `DEC 5000/200`. This method exploits the fact that the Newton polytopes for the last 4 equations are the same. Note that `HRB` requires about 1630 and 4730 seconds on these examples. Note also that

	kin1	kin2
CPU time	14.240	353.06
branching	89	5693
na-ee	17090	784687
na-te	10176	123032
na-tot	27266	907719
fe-ee	45656	1714779
fe-te	62080	854384
fe-tot	107736	2569163
pr-con	163	9505
proof	yes	yes

Table 10: `Newton` on the Kinematics Benchmarks

	4	5	6	7	8	9
CPU time	0.21	1.22	8.20	46.59	352.80	3311.42
growth		5.80	6.72	5.68	7.57	9.38
branching	24	119	517	2231	12248	82579
na-ee	834	4701	42481	214430	1622417	14031838
na-te	480	1976	6325	29238	157402	1219960
na-tot	1314	6677	48806	243668	1779819	15251798
fe-ee	2220	116628	99994	489894	3626847	30782836
fe-te	1293	6304	29825	160284	1062789	9118448
fe-tot	3513	17932	129819	650178	4689636	39901284
pr-con	37	147	687	2828	15265	104352
proof	yes	yes	yes	yes	yes	yes

Table 11: `Newton` on the economics modelling problem with initial intervals in $[-100, 100]$

`Newton` can prove the existence of solutions in the final intervals for these problems and that our computer representation uses intermediate variables

$$\begin{cases} x_{13} = x_4 + x_6 + x_8 \\ x_{14} = x_3 + x_5 + x_7 \\ x_{15} = x_4 + x_6 + x_8 \\ x_{16} = 3 * x_4 + 2 * x_6 + x_8 \end{cases}$$

to improve efficiently slightly in the first problem.

6.5 An Economics Modelling Application

The following example is taken from [24]. It is a difficult economic modelling problem that can be scaled up to arbitrary dimensions. For a given dimension n, the problem can be stated as the system

$$\begin{cases} (x_k + \sum_{i=1}^{n-k-1} x_i x_{i+k})x_n - c_k = 0 \quad (1 \leq k \leq n-1) \\ \sum_{l=1}^{n-1} x_l + 1 = 0 \end{cases}$$

and the constants can be chosen at random.

Table 11 reports the results for various values of n with an initial interval of $[-100, 100]$. It is interesting to compare those results with the continuation methods presented in [35]. [35] reports

	4	5	6	7	8	9
CPU time	0.60	3.35	22.53	127.65	915.24	8600.28
growth		5.58	6.72	5.66	7.16	9.39
branching	102	500	1778	7527	38638	244263
na-ee	2689	15227	122662	606805	4413150	36325819
na-te	978	3296	14055	64632	366436	2647016
na-tot	3667	18523	136717	671437	4779586	38972835
fe-ee	7140	38160	291206	1400216	9913850	80264897
fe-te	3288	15024	80110	429396	2825368	22672208
fe-tot	10428	53184	371716	1829612	12739218	102937105
pr-con	148	527	2080	8337	42704	271534
proof	yes	yes	yes	yes	yes	yes

Table 12: `Newton` on the economics modelling problem with initial intervals in $[-10^8, 10^8]$

times (on a DEC-5000/200) of about 1 second for $n = 4$, 6 seconds for $n = 5$, 50 seconds for $n = 6$ and 990 seconds for $n = 7$. `Newton` is substantially faster on this problem than this continuation method, since it takes about 47 seconds for $n = 7$. More importantly, the growth factor seems much lower in `Newton`. The continuation method has growths of about 8 and 20 when going from 5 to 6 and 6 to 7, while `Newton` has growths of about 6.72 and 5.68. Table 12 gives the same results for initial intervals in $[-10^8, 10^8]$. It is interesting to note that the computation times increase by less than a factor 3 and that the growth factor is independent of the initial intervals. Note also that Newton can establish the existence of solutions for these problems. Finally, it is worthwhile stating that the results were obtained for a computer representation where x_n has been eliminated in a problem of dimension n.

6.6 Combustion Application

This problem is also from Morgan's book [24] and represents a combustion problem for a tempara-ture of 3000°. The problem is described by the following sparse systems of equations

$$\begin{cases} x_2 + 2\ x_6 + x_9 + 2\ x_{10} = 10^{-5} \\ x_3 + x_8 = 3\ 10^{-5} \\ x_1 + x_3 + 2\ x_5 + 2\ x_8 + x_9 + x_{10} = 5\ 10^{-5} \\ x_4 + 2\ x_7 = 10^{-5} \\ 0.5140437\ 10^{-7}\ x_5 = x_1^2 \\ 0.1006932\ 10^{-6}\ x_6 = 2\ x_2^2 \\ 0.7816278\ 10^{-15}\ x_7 = x_4^2 \\ 0.1496236\ 10^{-6}\ x_8 = x_1\ x_3 \\ 0.6194411\ 10^{-7}\ x_9 = x_1\ x_2 \\ 0.2089296\ 10^{-14}\ x_{10} = x_1\ x_2^2 \end{cases}$$

which is typical of chemical equilibrium systems. Table 13 describes the results of `Newton` on for the initial intervals $[-1, 1]$ and $[-10^8, 10^8]$. `Newton` behaves well on this example, since the continuation method of [35] takes about 57 seconds. Note once again that a substantial increase in the size of the initial intervals only induces a slowdown of about 2.5 for `Newton`. Note also that `Newton` can prove the existence of the solutions and that we use a formulation where variable x_7 and x_3 have been eliminated.

	[-1,1]	[-10^8,10^8]
CPU time	4.06	9.94
branching	183	523
na-ee	10676	28473
na-te	5248	7952
na-tot	15924	36425
fe-ee	28928	77508
fe-te	22464	49632
fe-tot	51392	127140
pr-con	187	527
proof	yes	yes

Table 13: `Newton` on the Combustion Problem.

CPU time	*branching*	*na-ee*	*na-te*	*na-tot*	*fe-ee*	*fe-te*	*fe-tot*	*pr-con*	*proof*
6.32	256	13725	3400	17125	34811	17425	52236	425	yes

Table 14: `Newton` on the Chemestry Problem with Initial Intervals $[0, 10^8]$

6.7 Chemical Equilibrium Application

This problem originates from [17] and describes a chemical equilibrium system. The set of equations is as follows:

$$\left\{\begin{array}{l}
R = 10 \\
R_5 = 0.193 \\
R_6 = 0.002597/\sqrt{40} \\
R_7 = 0.003448/\sqrt{40} \\
R_8 = 0.00001799/40 \\
R_9 = 0.0002155/\sqrt{40} \\
R_{10} = 0.00003846/40 \\
x_1\ x_2 + x_1 - 3\ x_5 = 0 \\
2\ x_1\ x_2 + x_1 + x_2\ x_3^2 + R_8\ x_2 - R\ x_5 + 2\ R_{10}\ x_2^2 + R_7\ x_2\ x_3 + R_9\ x_2\ x_4 = 0 \\
2\ x_2\ x_3^2 + 2\ R_5\ x_3^2 - 8\ x_5 + R_6\ x_3 + R_7\ x_2\ x_3 = 0 \\
R_9\ x_2\ x_4 + 2\ x_4^2 - 4\ R\ x_5 = 0 \\
x_1\ x_2 + x_1 + R_{10}\ x_2^2 + x_2\ x_3^2 + R_8\ x_2 + R_5\ x_3^2 + x_4^2 - 1 + R_6\ x_3 + R_7\ x_2\ x_3 + R_9\ x_2\ x_4 = 0
\end{array}\right.$$

and all x_i's must be positive. The results are depicted in Table 14 for an initial interval $[0, 10^8]$. They indicate that `Newton` is particularly effective on this problem, since it takes about 6 seconds and proves the existence of a solution in the final intervals. Note that the continuation method of [35] takes about 56 seconds on this problem.

	$[-10, 10]$	$[-10^2, 10^2]$	$[-10^3, 10^3]$	$[-10^4, 10^4]$
CPU time	0.91	11.69	172.71	2007.51
growth		12.84	14.77	11.62
branching	52	663	9632	115377
na-ee	4290	57224	645951	6541038
na-te	810	6708	96888	1173456
na-tot	5100	63932	742839	7714494
fe-ee	11012	144843	1620538	16647442
fe-te	4104	48804	769056	9376632
fe-tot	15116	193647	2389594	26024074
pr-con	69	983	15980	195270
proof	yes	yes	yes	yes

Table 15: `Newton` on the Neurophysiology Problem.

6.8 A Neurophysiology Application

We conclude the experimental section by showing an example illustrating the limitations of `Newton`. The application is from neurophysiology [35] and consists of the following system of equations:

$$\left\{\begin{array}{l} x_1^2 + x_3^2 = 1 \\ x_2^2 + x_4^2 = 1 \\ x_5\ x_3^3 + x_6\ x_4^3 = c_1 \\ x_5\ x_1^3 + x_6\ x_2^3 = c_2 \\ x_5\ x_1\ x_3^2 + x_6\ x_4^2\ x_2 = c_3 \\ x_5\ x_1^2\ x_3 + x_6\ x_2^2\ x_4 = c_4 \end{array}\right.$$

No initial intervals for the variables were given and the constants c_i can be chosen at random. The continuation method of [35] solves this problem in about 6 seconds. The results of `Newton` are depicted in Table 15 for various initial intervals. `Newton` is fast when the initial intervals are small (i.e., $[-10, 10]$). Unfortunately, the running time of the algorithm increases linearly with the size of the initial intervals, showing a limitation of the method on this example.

7 Related Work and Discussion

The research described in this paper originated in an attempt to improve the efficiency of constraint logic programming languages based on intervals such as `BNR-Prolog` [26] and `CLP(BNR)` [3]. These Constraint Logic Programming languages use constraint solving as basic operation and they were based on the simple generalization of arc-consistency described previously, i.e.,

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

This approximation was enforced on simple constraints such as

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

and complex constraints were decomposed in terms of these simple constraints.

As mentioned earlier, this approach is not very effective [2] and our main goal was to design new approximations of arc-consistency that could make use of existing interval methods. The main problem was the difficulty in characterizing the pruning of the Newton operator N^* in a declarative way (in order to introduce it nicely in the above programming languages) and box-consistency emerged as an attempt to generalize the operator to make sure that the bounds of the interval were locally consistent. Subsequent research made us realize that box-consistency is independent of the Newton operator and can be enforced even if the functions are not continuous or differentiable. In addition, the value of applying box-consistency on several extensions became clear. On the one hand, box-consistency on the Taylor extension generalizes interval methods based on Gauss-Seidel iterations and enables us to capture nicely the Hansen-Segupta's operator. On the other hand, box-consistency on the natural and distributed extensions is really orthogonal to the pruning obtained from the Taylor expansion, producing a particularly effective algorithm. It is also worth pointing out that `Newton` spends most time in the natural and distributed extensions. However, for many applications, the use of the Taylor interval extension is critical to terminate the search quickly and to avoid generating many small intervals around the solutions. As a general observation, box-consistency on the natural and distributed extensions seem effective when far from a solution while box-consistency on the Taylor expansion seems effective when near a solution. It is worth mentioning that that the interval community has spent much effort to design additional techniques to speed up further the computation when near a solution but have not considered techniques to improve pruning when far from a solution.

It is interesting to note that the idea of using approximations of arc-consistency was also used independently by Hong and Stahl [11], who were also exposed to research on Constraint Logic Programming. Their use of projections is however quite different from ours. The key idea is to work with a set of boxes and to use projections to split a box into several subboxes by isolating all zeros of a projection. This gives an algorithm of a very different nature which cannot easily be characterized as a branch & prune algorithm since constraints are used to branch. Our approach seems to be more effective in practice, since their use of projections may generate many subboxes that may all need to be pruned away later on, implying much redundant work. Our approach postpones the branching until no pruning takes place and generates only subboxes when they are strictly necessary to progress. It is also very interesting to report that, on all benchmarks that we tested, the projection never contained more than two zeros. This seems to indicate that searching for all zeros may not be worthwhile in most cases and that box-consistency may be the right trade-off here. Finally, note that their approach seems to use implicitly an distributed extension[11] but they do not make use of the natural extension which is very important for some applications.

Note also that our current implementation does not use some of the novel techniques of the interval community such as the more advanced conditioners and splitting techniques of [13]. It is of course possible to include them easily, since the overall recursive structure of the implementations is essentially similar. Integrating these results would obviously be of benefit, since these techniques are complementary to ours.

The research described here also provides a uniform framework to integrate these techniques in Constraint Logic Programming, to understand the importance of the various pruning operators and their relationships and to suggest further research directions. For instance, higher notions of consistency such as path-consistency [18] may be worth investigating for some applications.

[11] The idea of sandwitching the interval function in between two real functions is described there.

8 Conclusion

In this paper, we presented a branch & prune algorithm to find all isolated solutions to a system of polynomial constraints over the reals. If the solution are not isolated, the algorithm will return boxes that contain several solutions. The algorithm is based on a single concept, box-consistency, which is an approximation of arc-consistency, a notion well-known in artificial intelligence. Box-consistency can be instantiated to produce the Hansen-Segupta's operator as well as other narrowing operators which are more effective when the computation is far from a solution. The algorithm and its mathematical foundations are simple. Moreover, the algorithm is shown to behave well on a variety of benchmarks from kinematics, mechanics, chemistry, combustion, and economics. It outperforms the interval methods we know of and compares well with continuation methods on their benchmarks. In addition, problems such as the Broyden banded function and the Moré-Cosnard discretization of a nonlinear integral equation can be solved for several hundred variables. Limitations of the method (e.g., a sensitivity to the size of the initial intervals on some problems) have also been identified.

Acknowledgments

We would like to thank the referees for their help in making the paper accessible to a wider audience, for pointing out several related works, and for their careful comments on the manuscript. We would like to thank F. Benhamou, A. Colmerauer, and B. Le Charlier for many interesting discussions on this topic. Pascal Van Hentenryck was supported in part by the Office of Naval Research under grant ONR Grant N00014-94-1-1153, the National Science Foundation under grant numbers CCR-9357704, a NSF National Young Investigator Award with matching funds of Hewlett-Packard. Deepak Kapur was supported in part by a grant from United State Air Force Office of Scientific Research AFOSR-91-0361.

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] R. Hammer, M. Hocks, M. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing I – Basic Numerical Problems, Theory, Algorithms, and PASCAL-XSC Programs.* Springer-Verlag, Heidelberg, 1993.

[5] E. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, New York, 1992.

[6] E.R. Hansen. Global Optimization Using Interval Analysis: the Multi-Dimensional Case. *Numer. Math*, 34:247–270, 1980.

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

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

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

[10] R.J. Hanson. Interval Arithmetic as a Closed Arithmetic System on a Computer. Jet Propulsion Laboratory Report 197, 1968.

[11] H. Hong and V. Stahl. Safe Starting Regions by Fixed Points and Tightening. *Computing*, 53(3-4):323–335, 1994.

[12] W.M. Kahan. A More Complete Interval Arithmetic. Lecture Notes for a Summer Course at the University of Michigan, 1968.

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

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

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

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

[17] K. Meintjes and A.P. Morgan. Chemical Equilibrium Systems as Numerical test Problems. *ACM Transactions on Mathematical Software*, 16:143–151, 1990.

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

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

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

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

[22] J.J. More and M.Y. Cosnard. Numerical Solution of Nonlinear Equations. *ACM Transactions on Mathematical Software*, 5:64–85, 1979.

[23] A.P. Morgan. Computing All Solutions To Polynomial Systems Using Homotopy Continuation. *Appl. Math. Comput.*, 24:115–138, 1987.

[24] A.P. Morgan. *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

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

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

[27] L.B. Rall. *Automatic Differentiation: Techniques and Applications.* Springer Lectures Notes in Computer Science, Springer Verlag, New York, 1981.

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

[29] S.M. Rump. Verification Methods for Dense and Sparse Systems of Equations. In J. (Ed.) Herzberger, editor, *Topics in Validated Computations*, pages 217–231. Elsevier, 1988.

[30] J. Siskind and D. McAllester. Nondeterministic Lisp as a Substrate for Constraint Logic Programming. In *AAAI-93*, pages 133–138, 1993.

[31] P. Van Hentenryck. A Logic Language for Combinatorial Optimization. *Annals of Operations Research*, 21:247–274, 1989.

[32] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series, The MIT Press, Cambridge, MA, 1989.

[33] P. Van Hentenryck, D. McAllester, and D. Kapur. Interval Methods Revisited. Technical Report No., MIT AI Lab, MIT, 1995.

[34] P. Van Hentenryck, V. Saraswat, and Y. Deville. The Design, Implementation, and Evaluation of the Constraint Language `cc(FD)`. In *Constraint Programming: Basics and Trends.* Springer Verlag, 1995.

[35] 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.