Newton: Constraint Programming over Nonlinear Real Constraints

Pascal Van Hentenryck and Laurent Michel

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-95-25
August 1995

Newton: Constraint Programming over Nonlinear Real Constraints

Pascal Van Hentenryck and Laurent Michel
Brown University, Box 1910
Providence, RI 02912
{pvh,ldm}@cs.brown.edu

Abstract

This paper is an introduction to Newton, a constraint programming language over nonlinear real constraints. Newton originates from an effort to reconcile the declarative nature of constraint logic programming (CLP) languages over intervals with advanced interval techniques developed in numerical analysis, such as the interval Newton method. Its key conceptual idea is to introduce the notion of box-consistency, which approximates arc-consistency, a notion well-known in artificial intelligence. Box-consistency achieves an effective pruning at a reasonable computation cost and generalizes some traditional interval operators. Newton has been applied to numerous applications in science and engineering, including nonlinear equation-solving, unconstrained optimization, and constrained optimization. It is competitive with continuation methods on their equation-solving benchmarks and outperforms the interval-based methods we are aware of on optimization problems.

1 Introduction

Many applications in science and engineering (e.g., chemistry, robotics, economics, mechanics) require finding all isolated solutions to a system of nonlinear real constraints or finding the minimum value of a nonlinear function subject to nonlinear constraints. These problems are difficult due to their inherent computational complexity (i.e., they are NP-hard) and due to the numerical issues involved to guarantee correctness (i.e., finding all solutions or the global optimum) and to ensure termination.

Newton is a constraint programming language designed to support this class of applications. It originates from an attempt to reconcile the declarative nature of CLP(Intervals) languages, such as BNR-Prolog, with advanced interval methods developed in numerical analysis, such as the interval Newton method.

Traditionally, CLP(Intervals) languages were designed in terms of simple primitive constraints (e.g. *add(x,y,z), mult(x,y,z), cos(x,z), ...*) on which they apply simple approximations of arc-consistency [18, 20], a notion well-known in artificial intelligence. Complex constraints are simply rewritten into a set of primitive constraints. The advantage of this methodology is the elegant operational semantics of the language. The inconvenience is that convergence may be slow and the pruning is relatively weak due to the decomposition process, making this approach unpractical on many applications.

By contrast, interval research in numerical analysis has focused, among other things, on producing reliable and reasonably fast methods to solve the above applications (e.g., [5, 6, 7, 8, 10, 12, 13, 14, 15, 21, 27, 34]). Many of these techniques use ideas behind Newton root finding method, exploit properties such as differentiability, and define various pruning operators, many of which extend the seminal work of Krawczyk [15]. These algorithms can often be viewed as an iteration

of two steps, constraint propagation and splitting, although they are rarely presented this way and it is not always clear what the constraint propagation step computes.

The key contribution of Newton is the notion of box-consistency, an approximation of arc-consistency which produces an effective tradeoff between precision and pruning. 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 been applied to numerous applications. It has been shown to be competitive on constraint-solving benchmarks with state-of-the-art continuation methods (e.g., [26, 38]) and to outperform traditional interval methods [37]. In addition, Newton has been applied to many benchmarks in global optimization (unconstrained and constrained optimization), outperforming interval methods we are aware of.

This paper is an introduction to Newton, a specification of its main functionalities, and a description of its performance on a number of benchmarks. The introduction illustrates the behaviour of Newton on a number of representative examples from univariate equation solving to multivariate constrained optimization. The specification contains static definitions of the main functionalities of Newton (e.g., constraint solving and optimization), as well as hints on how they are actually implemented. The performance results describe the efficiency of Newton on a number of traditional benchmarks from numerical analysis.

The rest of this paper is organized as follows. Section 2 describes the functionality of Newton on a number of applications to give readers an informal understanding of the language. Section 3 describes the language, i.e., its syntax and its semantics. The language is not fully covered. Rather we focus on the main features and we avoid entering into too much detail which would only be of interest to advanced users of the language. Section 4 describes the experimental results of Newton. Section 5 describes related work, while Section 6 concludes the paper.

2 A Short Tour of Newton

The purpose of this section is to illustrate the constraint-solving capabilities of Newton. We present examples which are simple to state and to read but readers should keep in mind that many problems require a complex phase of constraint generation that we do not discuss here. This part is of course performed in Newton as well, since Newton is in fact a superset of Prolog, but presents little interest. We begin with a simple program to compute the square root of a number.

```
squareRoot(X,Root) :-
     constraint [ Root >= 0 , Root^2 = X ].
```

The predicate squareRoot(X,Root) holds if Root is the square root of X. The query

```
?- squareRoot(2,Root)
```

returns the solution

```
Root in [1.41421356237306428127,1.41421356237312623172].
```

This simple program illustrates one of the fundamental aspects of Newton: intervals. Newton associates an interval with each variable and it is the role of constraints to narrow down these intervals to the required precision.[1] Consider now the problem of finding a solution to the equation

$$x^4 + x^2 = 1$$

in the interval $[0, 1]$. The Newton program

```
simpleRoot(X) :-
     constraint [ X in [0,1] , X^4 + X^2 = 1 ].
```

returns the solution

```
X in [0.78615137475949814493,0.78615138123592631648].
```

when queried with `?- simpleRoot(X)`. The same problem in the interval $[-1, 1]$ has two solutions. The Newton program which finds them both is as follows:

```
simpleRoots(X) :-
     constraint [ X in [-1,1] , X^4 + X^2 = 1 ],
     split(X).
```

The second goal of the clause is a nondeterministic predicate which splits the interval of X into smaller intervals until a solution to the constraint system is found. This goal is needed, since the constraint solver of Newton is incomplete. If omitted, the program would return

```
X in [-0.78615138123592631648,0.78615138123592631648]
```

for the query `?- simpleRoots(X)`. `split(X)` simply splits the interval associated with X into two parts and adds the resulting constraint nondeterministically. More precisely, Newton first assumes that X is in the left interval and constraint solving then produces immediately the solution

```
X in [-0.78615138123592631648,-0.78615137759960185270].
```

On backtracking, Newton assumes that X is in the right interval and generates the second solution that was shown previously.

This last example illustrates the traditional style of Newton programs: prune and branch. Constraints are used to prune the intervals associated with variables, while the branching nondeterministically splits an interval into two parts, whenever no more pruning is possible. To conclude this set of univariate examples, consider the search for the zeros of three related polynomials, an example which was used in a well-known textbook to illustrate the difficulty of nonlinear constraint solving. The Newton programs are simply

```
rootsF1(X) :-
     constraint [ X in [-10^8,10^8] , X^4 - 12 * X^3 + 47 * X^2 - 60 * X = 0],
     split(X).

rootsF2(X) :-
```

[1] All examples in this section assumes a default 8 digit precision.

```
     constraint [ X in [-10^8,10^8] , X^4 - 12 * X^3 + 47 * X^2 - 60 * X = -24],
     split(X).

rootsF3(X) :-
     constraint [ X in [-10^8,10^8] , X^4 - 12 * X^3 + 47 * X^2 - 60 * X = -24.1],
     split(X).
```

The query `?- rootsF1(X)` returns the four solutions

```
X in [-0.00000000000000122569,0.00000000000000014497];
X in [2.99999999999997202237,3.00000000000016298075];
X in [3.99999999999988498089,4.00000000000130828682];
X in [4.99999999999923971927,5.00000000000264588352].
```

The query `?- rootsF2(X)` returns the two solutions

```
X in [0.88830577862955317769,0.88830577912592989521];
X in [0.99999999976380837818,1.00000000000002575718].
```

The query `?- rootsF3(X)` fails, indicating the absence of zeros. The above examples take a couple of milliseconds in `Newton`.

We now turn to multivariate examples and we consider first the intersection of a circle and of a parabola. The `Newton` program is as follows:

```
circleAndParabola(L) :-
     L = [X1,X2],
     constraint [
          X1^2 + X2^2 = 1,
          X1^2 - X2 = 0
     ],
     listsplit(L).
```

It follows the structure of the previous programs, except that the predicate `listsplit` is used instead of `split`, since there are several variables. If the predicate `listsplit` is omitted, the program returns the box

```
X in [-1.00000000029311686412,1.00000000029311686412]
Y in [-0.00000000000000000000,1.00000000029311686412].
```

Splitting once on variable `X1` leads directly to the first solution

```
X in [-0.78615137775742405247,-0.78615137775742249814]
Y in [0.61803398874989357025,0.61803398874989623480].
```

On backtracking, `Newton` produces the second solution

```
X in [0.78615137775742249814,0.78615137775742405247]
Y in [0.61803398874989345923,0.61803398874989623480].
```

```
robotAngles(L) :-
     L = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12],
     constraint [
          X1^2 + X2^2 = 1,
          X3^2 + X4^2 = 1,
          X5^2 + X6^2 = 1,
          X7^2 + X8^2 = 1,
          X9^2 + X10^2 = 1,
          X11^2 + X12^2 = 1,
          X12 * (X4 + X6 + X8) - (X10 * X11 * (X3 + X5 + X7)) = 0.4077,
          X2 * X9 * (X4 + X6 + X8) + X1 * X10 = 1.9115,
          X9 * (X3 + X5 + X7) = 1.9791,
          X2 * (3 * X4 + 2 * X6 + X8) = 4.0616,
          X1 * (3 * X4 + 2 * X6 + X8) = 1.7172,
          3 * X3 + 2 * X5 + X7 = 3.9701
     ],
     listsplit(L).
```

Figure 1: A Newton Program for Robot Kinematics.

A more interesting application is robot kinematics, where the goal is to find the angles for the joints of a robot arm so that the robot hand ends up in a specified position. A Newton program, solving a problem given in [10], is depicted in Figure 1. Once again, the program follows the simple structure that we have encountered so far. It takes about 15 seconds to find all solutions on a SUN Sparc-10 workstation.

Nonlinear constraint techniques are also of primary importance to solve chemical equilibrium systems. Our next example is an equilibrium system to describe the propulsion of propane in air and is taken from [19]. The Newton program which finds the solution in about 5 seconds is depicted in Figure 2.

The above programs guarantee that any solution to the constraint system are located in at least one box returned by Newton. Hence, we can safely conclude (modulo implementation bugs) that there is no solution when Newton fails. However, there is no guarantee in the above programs that a box actually contains a solution. The problem of existence and unicity of solutions is in fact a difficult problem, which is an active research area in interval methods. Newton provides a facility to obtain safe boxes, i.e., boxes that are guaranteed to contain a single solution. The proof of existence and unicity is obtained numerically during the computation. For instance, the first univariate polynomial problem described previously can be stated as follows to obtain safe boxes:

```
rootsF1(S) :-
     S is_a_Sbox_of [ X in [-10^8,10^8] , X^4 - 12 * X^3 + 47 * X^2 - 60 * X = 0].
```

The query `?- rootsF1(X)` returns the four solutions

```
X in [-0.00000000000000122569,0.00000000000000014497];
X in [2.99999999999997202237,3.00000000000016298075];
```

```
equilibrium(L) :-
     L = [Y1,Y2,Y3,Y4,Y5],

     R = 10,
     R5 = 0.193,
     R6 = 0.002597,
     R7 = 0.003448,
     R8 = 0.00001799 * 0.025,
     R9 = 0.0002155,
     R10 = 0.00003846 * 0.025,

     constraint [
          Y1 in [0,10^8],
          Y2 in [0,10^8],
          Y3 in [0,10^8],
          Y4 in [0,10^8],
          Y5 in [0,10^8],

          sqrt(40) * (3 * (2 * Y1 * Y2 + Y1 + Y2 * Y3^2 + R8 * Y2) -
          R * Y5 + 6 * R10 * Y2^2) + 3 * (R7 * Y2 * Y3 + R9 * Y2 * Y4) = 0,

          sqrt(40) * (3 * (2 * Y2 * Y3^2 + 2 * R5 * Y3^2) - 8 * Y5) +
          3 * (R6 * Y3 + R7 * Y2 * Y3) = 0,

          3 * R9 * Y2 * Y4 + sqrt(40) * (6 * Y4^2 - 4 * R * Y5) = 0,

          sqrt(40) * (Y1 * Y2 + Y1 + R10 * Y2^2 + Y2 * Y3^2 + R8 * Y2 +
          R5 * Y3^2 + Y4^2 - 1) + R6 * Y3 + R7 * Y2 * Y3 + R9 * Y2 * Y4 = 0,

          Y5 = Y1 * (Y2 + 1)
     ],
     listsplit(L).
```

Figure 2: A Newton Program for the Combustion of Propane in Air.

```
X in [3.99999999999988498089,4.00000000000130828682];
X in [4.99999999999923971927,5.00000000000264588352].
```

indicating that there is a unique solution in each box. The proof of existence and unicity cannot always be obtained, even if the box actually contains a single solution. `Newton` does not return the box as a solution in this case. For all examples described so far, there is no difficulty in obtaining safe boxes. Note also that no splitting goal is necessary when safe boxes are requested, since `Newton` needs to control the splitting process when a safe box is obtained. For symmetry, `Newton` also provides a construction for obtaining boxes which are not necessarily safe and the above program can simply be written as follows with this facility:

```
rootsF1(S) :-
     S is_a_Cbox_of [ X in [-10^8,10^8] , X^4 - 12 * X^3 + 47 * X^2 - 60 * X = 0].
```

The "`C`" stands for canonical and its meaning will become clear later on.

In addition to solving systems of linear constraints, `Newton` has also been instrumental in solving global optimization problems. The solving of global optimization problems are in fact very similar to solving systems of constraints and `Newton` employs mostly the same techniques to solve them. Our first example in this area is an unconstrained minimization, i.e., the search for the minimum value of a nonlinear function. A typical problem taken from [17] is given by the minimization of the function

$$\prod_{k=1}^{2}\sum_{i=1}^{5} i\cos((i+1)x_k + i).$$

Restricting attention to the range $[-10, 10]$ for the variables, a simple `Newton` program to solve this problem is as follows:

```
minimization(Min,Boxes) :-
     minof
        (cos(2*X1+1)+2*cos(3*X1+2)+3*cos(4*X1+3)+4*cos(5*X1+4)+5*cos(6*X1+5)) *
        (cos(2*X2+1)+2*cos(3*X2+2)+3*cos(4*X2+3)+4*cos(5*X2+4)+5*cos(6*X2+5))
     in
        [-10,10]
     isin
        Min
     for Boxes.
```

For variable `Min`, `Newton` returns the interval

```
Min in [-186.7309211408969247,-186.7308966622586581]
```

which bounds the value of the global optimum and, for variable `Boxes`, a list of the global solutions. There are 18 global optima in this problem, one of them being

```
X1 in [5.4828642061208122,5.4828642089241431]
X2 in [4.8580568744302264,4.8580568843370485].
```

```
minimization(Min,Boxes) :-
     B1 = 4.97,
     B2 = -1.88,
     B3 = -29.08,
     B4 = -78.02,

     minof
        4.3 * X1 + 31.8 * X2 + 63.3 * X3 + 15.8 * X4 + 68.5 * X5 + 4.7 * X6
     subject_to [
        X1 in [0,0.31],
        X2 in [0,0.046],
        X3 in [0,0.068],
        X4 in [0,0.042],
        X5 in [0,0.028],
        X6 in [0,0.0134],
        17.1 * X1 + 38.2 * X2 + 204.2 * X3 + 212.3 * X4 + 623.4 * X5 +
        1495.5 * X6 - 169 * X1 * X3 - 3580 * X3 * X5 - 3810 * X4 * X5
        18500 * X4 * X6 - 24300 * X5 * X6 >= B1,

        17.9 * X1 + 36.8 * X2 + 113.9 * X3 + 169.7 * X4 + 337.8 * X5 +
        1385.2 * X6 - 139 * X1 * X3 - 2450 * X4 * X5 - 16600 * X4 * X6 -
        17200 * X5 * X6 >= B2,

        -273 * X2 - 70 * X4 - 819 * X5 + 26000 * X4 * X5 >= B3,

        159.9 * X1 - 311 * X2 + 587 * X4 + 391 * X5 + 2198 * X6
        - 14000 * X1 * X6 >= B4 ]
     isin
        Min
     for Boxes.
```

Figure 3: A Newton Program for Constrained Optimization.

Newton takes about 20 seconds to solve this problem. Note that Newton returns the global optimum, not a local optimum.

Newton has also solved a variety of constrained optimization problems. A beautiful example taken from [9] is shown in Figure 3. Newton solves this problem in about 2 seconds. Newton has two primitives for constrained optimization, minof and safe_minof. They differ on the way they compute the upper bound to the minimum value (and hence the optimum solutions). In minof, the upper bound is computed using canonical boxes which make the constraint system box-consistent. As a consequence, the result may not be an upper-approximation of the minimum value, since these boxes are not guaranteed to contain solutions. In contrast, safe_minof uses safe boxes of the constraint system to compute the upper bound which is then a conservative (but possibly poor) approximation of the upper bound. This is explained in more detail in Section 3.6.

3 The Constraint Language Newton

This section describes the specification of Newton in more detail. It describes successively its syntax, its constraint solver, its splitting operations, its facilities to obtain safe and canonical boxes, its primitives for unconstrained and constrained optimization, and some pragmatic issues.

3.1 The Syntax of Newton

Figure 4 describes an outline of the syntax of Newton. The syntax is not complete and omits certain features of the language (besides the obvious omissions) but it is representative of the kernel of the implementation. Newton is a superset of Prolog. A program is a set of clauses, each of which is either composed of a head or of a head and a body. A body is either a constraint, a goal, or a conjunction of two bodies. A constraint is either a single basic constraint or list of basic constraints. Basic constraints are equations and inequalities over expressions as well as membership constraints which are abbreviations for a conjunction of two inequalities. Expressions are constructed from variables, floating-point numbers, multiplication, addition, subtraction, exponentiation by a natural number, sine and cosine, and the logarithm and exponentiation functions. A goal is either an atom, or one of the facilities available in the language for safe boxes and optimization.

3.2 Constraint Solving

We now turn to the study of nonlinear constraint solving in Newton. Since Newton is a superset of Prolog, it also includes equations over finite trees but these are not covered in this article. The rest of this section is organized as follows. Section 3.2.1 briefly reviews some fundamental concepts of interval arithmetic, Section 3.2.2 discusses the representation of constraints, Section 3.2.3 introduces the notion of box-consistency, Section 3.2.4 describes several interval extensions on which Newton applies box-consistency, Section 3.2.5 describes the specification of the constraint solver. The internal details of the solver are not covered here. Instead, we focus on a high-level "static" specification of the language. The reader interested in the details of the solver can consult [37].

```
⟨Program⟩     ::=   ⟨Clauses⟩

⟨Clauses ⟩    ::=   ⟨Head⟩.                                                    |
                    ⟨Head⟩ :- ⟨Body⟩.                                          |
                    ⟨Clauses⟩ ⟨Clauses⟩

⟨Head⟩        ::=   ⟨Atom⟩

⟨Body⟩        ::=   ⟨Constraint⟩                                               |
                    ⟨Goal⟩                                                     |
                    ⟨Body⟩ , ⟨Body⟩

⟨Constraint⟩  ::=   ⟨C⟩                                                        |
                    [⟨C⟩]*

⟨C⟩           ::=   ⟨Expr⟩ >= ⟨Expr⟩                                           |
                    ⟨Expr⟩ <= ⟨Expr⟩                                           |
                    ⟨Expr⟩ =  ⟨Expr⟩                                           |
                    ⟨Expr⟩ in  ⟨Range⟩

⟨Expr⟩        ::=   ⟨Var⟩                                                      |
                    ⟨Float⟩                                                    |
                    ⟨Expr⟩ * ⟨Expr⟩                                            |
                    ⟨Expr⟩ + ⟨Expr⟩                                            |
                    ⟨Expr⟩ - ⟨Expr⟩                                            |
                    ⟨Expr⟩ ^ ⟨Nat⟩                                             |
                    - ⟨Expr⟩                                                   |
                    sin(⟨Expr⟩)                                                |
                    cos(⟨Expr⟩)                                                |
                    log(⟨Expr⟩)                                                |
                    exp(⟨Expr⟩)                                                |

⟨Range⟩       ::=   [⟨Expr⟩ , ⟨Expr⟩]

⟨Goal⟩        ::=   ⟨Atom⟩                                                     |
                    ⟨Var⟩ is_a_Sbox_of ⟨Constraint⟩                            |
                    ⟨Var⟩ is_a_Cbox_of ⟨Constraint⟩                            |
                    minof ⟨Expr⟩ in ⟨Range⟩ is_in ⟨Var⟩ for ⟨Var⟩               |
                    minof ⟨Expr⟩ subject_to ⟨Constraint⟩ is_in ⟨Var⟩ for ⟨Var⟩  |
                    safe_minof ⟨Expr⟩ subject_to ⟨Constraint⟩ is_in ⟨Var⟩ for ⟨Var⟩
```

Figure 4: An Outline of the Syntax of Newton.

3.2.1 Interval Arithmetic

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

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

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

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 $\langle r_1, \ldots, r_n \rangle$. A canonical interval is an interval of the form $[l, l]$ or of the form $[l, l^+]$. A canonical box is a tuple of canonical intervals. $\mathcal{Q}$ is the set of rational numbers and $\mathcal{N}$ is the set of natural numbers. Finally, we use the following notations.

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

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

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.

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

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.

3.2.2 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. We will abuse notation by denoting functions (resp. constraints) and their representations by the same symbol. In this section, 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 interval 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. For convenience, we also assume in the rest of this section that all constraints are defined over variables $x_1, \ldots, x_n$.

3.2.3 Box Consistency

Box-consistency [1] is an approximation of arc-consistency, a notion well-known in artificial intelligence [18] 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 [29, 3], is easily enforced on simple constraints such as

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

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 [1] for experimental results on this approach and their comparison with the approach presented in this paper.

The notion of box-consistency introduced in [1] 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 $l = left(I_i)$ and $u = right(I_i)$ 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. 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.

3.2.4 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.

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

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. A very nice application where this extension is fundamental is the Moré-Cosnard discretization of a nonlinear integral equation described in the experimental results. Using the natural extension allows users to minimize the problem of dependency of interval arithmetic and hence to increase precision.

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.[3] 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

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

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

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.[4]

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 $X_1X_2 + X_1X_3 = \overline{0}$.

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 [21] 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.

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 [31].

3.2.5 Specification of the Constraint Solver

We are now in position to specify the behaviour of the constraint solver of `Newton`. Informally speaking, `Newton` enforces box-consistency on the three interval extensions of the constraints without removing any solution.

Definition 11 Let $\mathcal{S}$ be a system of constraints $\{c_1, \ldots, c_n\}$ and $\vec{I}$ be a box. $\mathcal{S}$ is box-consistent wrt $\vec{I}$ if

$$\{\widehat{c_1}, \ldots, \widehat{c_n}\} \cup \{\widetilde{c_1}, \ldots, \widetilde{c_n}\} \cup \{c_1^{t(\vec{I})}, \ldots, c_n^{t(\vec{I})}\}$$

is box-consistent wrt $\vec{I}$.

Specification 1 [Constraint Solving] Let $\mathcal{S}$ be a system of constraints. The constraint solver of `Newton` returns a box $\vec{I}$ which satisfies the following two properties:

- $\mathcal{S}$ is box-consistent wrt $\vec{I}$;

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

- if $\langle r_1, \ldots, r_n \rangle$ is a solution of $\mathcal{S}$, $\langle r_1, \ldots, r_n \rangle \in \vec{I}$.

Note that there may be more than one box $\vec{I}$ satisfying the above conditions due to the nature of the Taylor extension. However, any of these boxes is sound by definition and which box is actually returned is left as an implementation issue.

3.3 Splitting

As should be clear from the above description, the constraint solver in `Newton` is incomplete. To solve practical applications, it is necessary to enhance the constraint solver with nondeterministic choices. `Newton` supports this process by providing two splitting operations which can be specified as follows.

Specification 2 [Splitting Operations] Procedure `split(t)` holds iff `t` is a canonical interval. Procedure `listSplit([`$\mathtt{t}_1$`,...,`$\mathtt{t}_n$`)])` holds iff $\langle \mathtt{t}_1, \ldots, \mathtt{t}_n \rangle$ is a canonical box.

In its current implementation, `Newton` does not simply enumerate all the canonical intervals. Rather, it performs a binary search, splitting the intervals in two parts until canonical boxes are obtained. This implementation gives naturally rise to a branch and prune algorithm, which alternates a constraint propagation phase and nondeterministic choices. In the case of `listSplit`, `Newton` uses a simple round-robin strategy to split the box.

3.4 Safe Boxes

`Newton` provides facilities to guarantee that boxes returned as answers be safe, i.e., that they actually contain a unique solution to the constraint system. Verifying that a box $\vec{I}$ provably satisfies an inequality $f \geq 0$ is relatively easy. It is sufficient to evaluate f over $\vec{I}$, producing an interval $[l, u]$. The constraint is provably satisfied if $l \geq 0$. Determining if a box provably satisfies an equation or a system of equations is obviously more difficult and this is an active research area in interval analysis. The techniques currently used in `Newton` are described in [37]. We now specify the facilities to obtain safe and canonical boxes in `Newton`.

Specification 3 [Safe and Canonical Boxes] The procedure

```
Box is_a_Sbox_of System
```

holds iff `System` is box-consistent wrt `Box` and `Box` is a safe box of `System`. The procedure

```
Box is_a_Cbox_of System
```

holds iff `System` is box-consistent wrt `Box` and `Box` is a canonical box.

3.5 Unconstrained Optimization

We now turn to unconstrained optimization in `Newton`. Informally speaking, the procedure

```
minof F in Range is_in Min for Solutions
```

holds if `Min` is the (global) minimum value of `F` (which is assumed to be in `Range`) and `Solutions` is a list of all (global) minima. The precise specification requires however a number of definitions.

Definition 12 [Interval Minimum] Let K be the set of canonical boxes of the form $\langle I_1, \ldots, I_n \rangle$. The interval-minimum value of a function f, denoted by *imin(f)*, is an interval $[l, u]$ such that

$$l = \min_{\vec{I} \in K} \mathit{left}(\widehat{f}(\vec{I}))$$
$$u = \min_{\vec{I} \in K} \mathit{right}(\widehat{f}(\vec{I}))$$

An interval-minimum of f is a canonical box $\vec{I}$ such that

$$f(\vec{I}) \cap \mathit{imin}(f) \neq \emptyset.$$

Specification 4 [Unconstrained Minimization] The procedure

```
minof F in Range is_in Min for Boxes
```

holds iff

- `Min` is the interval-minimum value of `F`;
- `Boxes` is a list of interval-minima of `F` in `range` such that each (global) minimum of `F` belongs to at least one box of `Boxes`.

The current implementation of `Newton` is based on a depth-first branch and bound which uses partial derivatives of the objective function to speed up the search for the minima.

3.6 Constrained Optimization

Constrained optimization in `Newton` looks superficially the same as unconstrained optimization. However, special care should be exercised when guaranteed bounds are required. Informally speaking, the procedure

```
minof F subject_to Constraints is_in Min for Solutions
```

holds if `Min` is the (global) minimum value of `F` satisfying the constraint `Constraints` and `Solutions` is a list of all (global) minima. The precise specification requires a number of definitions, closely related to those of the previous section.

Definition 13 [Interval Minimum] Let f be a function, $\mathcal{S}$ be a system of constraints, and let K_b be the set of canonical boxes $\vec{I}$ such that $\mathcal{S}$ is box-consistent wrt $\vec{I}$. The interval-minimum value of a function f wrt $\mathcal{S}$, denoted by *imin(f,$\mathcal{S}$)*, is an interval $[l, u]$ such that

$$l = \min_{\vec{I} \in K_b} \mathit{left}(\widehat{f}(\vec{I}))$$
$$u = \min_{\vec{I} \in K_b} \mathit{right}(\widehat{f}(\vec{I}))$$

An interval-minimum of f wrt $\mathcal{S}$ is a canonical box $\vec{I}$ of K_b such that

$$f(\vec{I}) \cap \mathit{imin}(f, \mathcal{S}) \neq \emptyset.$$

Specification 5 [Constrained Minimization] The procedure

```
minof F subject_to Constraints is_in Min for Boxes
```

holds iff

- `Min` is *imin*(`F`,`Constraints`);
- `Boxes` is a list of interval-minima of `F` wrt `Constraints` such that each (global) minimum of `F` wrt `Constraints` belongs to at least one box of `Boxes`.

The above procedure does not necessarily provide guaranteed bounds on the minimum value of the function wrt the constraints. The problem comes from the fact that box-consistent canonical boxes are not guaranteed to contain any solution to the constraint system. As a consequence, the upper bound which is returned may not be an upper approximation of the minimum value. To remedy this problem, `Newton` provides a facility to obtain guaranteed bounds as well. The specification is slightly more involved.

Definition 14 [Safe Interval Minimum] Let f be a function, $\mathcal{S}$ be a system of constraints, K_b be the set of canonical boxes $\vec{I}$ such that $\mathcal{S}$ is box-consistent wrt $\vec{I}$, and K_s be the set of safe boxes $\vec{I}_s$ such that $\mathcal{S}$ is box-consistent wrt $\vec{I}_s$. The safe interval-minimum value of a function f wrt $\mathcal{S}$, denoted by *smin*(f,$\mathcal{S}$), is an interval $[l, u]$ such that

$$l = \min_{\vec{I} \in K_b} left(\widehat{f}(\vec{I}))$$
$$u = \min_{\vec{I} \in K_s} right(\widehat{f}(\vec{I}))$$

A conservative interval-minimum of f wrt $\mathcal{S}$ is a canonical box $\vec{I}$ of K_b such that

$$f(\vec{I}) \ \cap \ smin(f, \mathcal{S}) \ \neq \ \emptyset.$$

The main difference is of course the computation of the upper bound, which uses safe boxes instead of canonical boxes.

Specification 6 [Safe Constrained Minimization] The procedure

```
safe_minof F subject_to Constraints is_in Min for Boxes
```

holds iff

- `Min` is *smin*(`F`,`Constraints`);
- `Boxes` is a list of conservative interval-minima of `F` wrt `Constraints` such that each (global) minimum of `F` wrt `Constraints` belongs to at least one box of `Boxes`.

The current implementation of `Newton` enforces box-consistency on the Fritz-John conditions to find the minima.

3.7 Pragmatics

`Newton` has also a number of pragmas that affect the semantics of the programs. Perhaps the most important one is the fact that users can specify the precision of the system, since they are rarely interested in canonical boxes. In practice, a precision of, say, 10^{-8} may be acceptable and users can specify that `Newton` returns boxes whose width is smaller than 10^{-8}. The semantics of the language essentially remains the same, except that the system returns boxes of the required precision. These boxes covered all the boxes that would be returned otherwise. Another feature of `Newton` is that the constraint solver may use (and typically uses) linear combinations of constraints to improve pruning. Hence box-consistency is applied to a system which is different from the system specified by the program.

4 Experimental Results

This section describes some experimental results of `Newton` on traditional benchmarks. It considers successively equation solving, unconstrained optimization, and constrained optimization.

4.1 Equation Solving

This section reports experimental results of `Newton` on a variety of standard benchmarks for equation solving. The benchmarks were taken from papers on numerical analysis [24], interval analysis [8, 10, 23], and continuation methods [38, 26, 25, 19]. 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.[5] Finally, we compare `Newton` with a state-of-the-art continuation method [38], 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}. 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 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 this section describes each benchmark. Note that `Newton` solves `Broyden`, `Moré-Cosnard`, and interval benchmarks `i1, i2, i3` and `i5` without backtracking (contrary to most interval methods we know of). `Newton` is essentially linear on `Broyden` and quadratic on `Moré-Cosnard`.

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

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

where $J_i = \{j \mid j \neq i \ \& \ \max(1, i-5) \leq j \leq \min(n, i+1)\}$. 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 basis to compare various methods.

Discretization of a Nonlinear Integral Equation This example comes from [24] 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 [32].

[5] 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 on Equation Solving.

Interval Arithmetic Benchmarks These are traditional benchmarks from interval arithmetic papers [22, 10]. Benchmark `i1` is the following set of equations

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

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

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

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

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

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

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

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

Kinematics Applications Application `kin1` comes from robotics and describes the inverse kinematics of an elbow manipulator [10]. It consists of a sparse system with 12 variables and the set of equations is as follows:

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

The second benchmark, denoted by `kin2`, is from [25] 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:

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

where the coefficients a_{ki} are given in table 2. In both examples, the initial intervals were given as $[-10^8, 10^8]$.

An Economics Modelling Application The following example is taken from [26]. 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

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

and the constants can be chosen at random.

- 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 2: Coefficients for the Inverse Kinematics Example.

Combustion Application This problem is also from Morgan's book [26] and represents a combustion problem for a temperature 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.

Chemical Equilibrium Application This problem originates from [19] and describes a chemical equilibrium system. The set of equations is as follows:

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

and all x_i's must be positive.

A Neurophysiology Application This example illustrates the limitations of `Newton`. The application is from neurophysiology [38] and consists of the following system of equations:

$$\begin{cases} x_1^2 + x_3^2 = 1 \\ x_2^2 + x_4^2 = 1 \\ x_5\ x_3^3 + x_6\ x_4^3 = c_1 \\ x_5\ x_1^3 + x_6\ x_2^3 = c_2 \\ x_5\ x_1\ x_3^2 + x_6\ x_4^2\ x_2 = c_3 \\ x_5\ x_1^2\ x_3 + x_6\ x_2^2\ x_4 = c_4 \end{cases}$$

No initial intervals for the variables were given and the constants c_i can be chosen at random.

4.2 Unconstrained Optimization

Table 3 describes the results of `Newton` on unconstrained optimization. The benchmarks were taken mainly from [17, 11, 33, 35] and, for each of them, we give the number of variables, the range of the variables, the `CPU` time, and the number of splits. The experimental results once again exhibit a number of interesting facts. `Newton` is able to solve problems such as `Levy5` and `Levy6` in essentially linear time in the number of variables. `Newton` solves the problems `Ratz25`, `Ratz27`, and `Ratz210` without splitting. These problems were used in [33] to study splitting strategies. Finally, `Newton` does not exhibit the behaviour of traditional interval methods on problems such as the Rosenbrock function. The performance of the traditional interval degrades substantially when the initial intervals are large, while `Newton` can be used with arbitrarily large intervals in these cases (without degrading the performance. We now describe each benchmark in detail.

`Hump` is the three-hump camel function

$$f(x_1, x_2) = 12x_1^2 - 6.3x_1^4 + x_1^6 + 6x_2(x_2 - x_1).$$

Benchmarks	v	Range	Time	Splits
Hump	2	$[-10^7, 10^8]$	0.17	3
Levy1	1	$[-10^7, 10^7]$	0.09	2
Levy2	1	$[-10, 10]$	0.61	4
Levy3	2	$[-10, 10]$	16.14	30
Levy4	2	$[-10, 10]$	2.13	7
Levy5	3	$[-10, 10]$	0.75	1
Levy5	5	$[-10, 10]$	1.45	1
Levy5	10	$[-10, 10]$	4.35	1
Levy5	20	$[-10, 10]$	15.27	1
Levy5	40	$[-10, 10]$	59.08	1
Levy5	80	$[-10, 10]$	235.22	1
Levy6	3	$[-10, 10]$	0.96	1
Levy6	5	$[-10, 10]$	1.57	1
Levy6	10	$[-10, 10]$	4.29	1
Levy6	20	$[-10, 10]$	14.86	1
Levy6	40	$[-10, 10]$	64.11	1
Levy6	80	$[-10, 10]$	372.39	1
Beale	2	$[-4.5, 4.5]$	2.50	3
Beale	2	$[-10^2, 10^2]$	3.31	12
Beale	2	$[-10^4, 10^4]$	5.52	31
Beale	2	$[-10^7, 10^7]$	23.29	61
Schwefel1	3	$[-10^7, 10^7]$	0.24	0
Booth	2	$[-10^7, 10^7]$	0.11	0
Powell	4	$[-10, 20]$	6.69	267
Schwefel3	2	$[-10^7, 10^7]$	0.03	0
Rosenbrock	2	$[-10^7, 10^7]$	0.33	10
Ratz1	5	$[-500, 600]$	1.19	0
Ratz25	4	$[0, 10]$	2.86	0
Ratz27	4	$[0, 10]$	4.44	0
Ratz210	4	$[0, 10]$	7.42	0
Ratz3	6	$[0, 1]$	9.13	2
More1	3	$[-4, 4]$	10.04	5
More2	4	$[-25, 25]$	189.56	32

Table 3: Summary of the Experimental Results on Unconstrained Optimization.

`Levy1` is the function

$$f(x) = x^6 - 15x^4 + 27x^2 + 250.$$

`Levy2` is the function

$$f(x) = -\sum_{i=1}^{5} i\cos[(i+1)x + i].$$

`Levy3` is the function

$$f(x_1, x_2) = \prod_{k=1}^{2}\sum_{i=1}^{5} i\cos((i+1)x_k + i).$$

This function has 760 local minima and 18 global minima in [-10,10].

`Levy4` is the related function

$$\prod_{k=1}^{2}\sum_{i=1}^{5} i\cos((i+1)x_k + i) + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2.$$

`Levy5` is the function

$$f(x_1, \ldots, x_n) = \sin(\pi y_1)^2 + \sum_{i=1}^{n-1}(y_i - 1)^2(1 + 10\sin(\pi y_{i+1})^2) + (y_n - 1)^2$$

where

$$y_i = 1 + (x_i - 1)/4 \quad (1 \le i \le n).$$

`Levy6` is the function

$$f(x_1, \ldots, x_n) = \sin(3\pi x_1)^2 + \sum_{i=1}^{n-1}(x_i - 1)^2(1 + 10\sin(3\pi x_{i+1})^2) + (x_n - 1)(1 + \sin(2\pi x_n)).$$

For $n = 2$, this problem has 900 local minima when variables range over $[-10, 10]$. For $n = 3$ and $n = 4$, the number of local minima goes up to 2700 and 71000.

`Beale` is the function

$$f(x_1, x_2) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x_2^2))^2 + (2.625 - x_1(1 - x_2^3))^2.$$

`Schwefel1` is the function

$$f(x_1, x_2, x_3) = \sum_{i=1} 3(x_1 - x_i^2)^2 + (X_i - 1)^2.$$

`Booth` is the function

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2.$$

`Schwefel2` is the function

$$f(x_1, x_2, x_3) = \sum_{k=1}^{10} F_k^2$$

where

$$F_k = \exp(-0.1kx_1) - \exp(-0.1kx_2) - A_k x_3$$
$$A_k = \exp(-0.1k) - \exp(-k)$$

`Powell` is the function

$$(x_1 + 10 * x_2)^2 + 5 * (x_3 - x_4)^2 + (x_2 - 2 * x_3)^4 + 10 * (x_1 - x_4)^4$$

It is a difficult problem for interval methods in general, since the Hessian is singular at the solution point.

`Schwefel3` is the function

$$f(x_1, x_2, x_3) = \sum_{i=2}^{3} [x_1 - x_i^2)^2 + (1 - x_i)^2].$$

`Rosenbrock` is the function

$$f(x_1, x_2) = 100(x_2 - x_1)^2 + (x_1 - 1)^2.$$

Interval methods are usually very dependent on the size of the initial box for this problem, although this is not the case for `Newton`.

`Ratz1` is the function

$$f(x_1, \ldots, x_5) = \sum_{i=1}^{5} x_i^2/400 - \prod_{i=1} 5\cos(x_i/\sqrt{i}) + 1.$$

`Ratz25, Ratz27` **and** `Ratz210` are generated from the function

$$f(x_1, \ldots, x_m) = -\sum_{i=1}^{m} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

for m equal to $5, 7$ and 10 respectively. The matrix A and the vector c are defined as follows:

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}$$

`Ratz3` is the function

$$f(x_1,\ldots,x_6) = -\sum_{i=1}^{4} c_i \exp(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2)$$

where $A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$, $C = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}$ and

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

`More1` is the function

$$f(x_1,\ldots,x_3) = \sum_{i=1}^{15} (x_1 \exp(-0.5x_2(t_i - x_3)^2) - y_i)^2$$

where $t_i = (8-i)/2$ and

i	y_i
$1, 15$	0.0009
$2, 14$	0.0044
$3, 13$	0.0175
$4, 12$	0.0540
$5, 11$	0.1295
$6, 10$	0.2420
$7, 9$	0.3521
8	0.3989

`More2` is the function

$$f(x_1,\ldots,x_4) = \sum_{i=1}^{20} [(x_1 + t_i x_2 - \exp(t_i))^2 + (x_3 + x_4 \sin(t_i) - \cos(t_i))^2]^2$$

where $t_i = i/5$.

4.3 Constrained Optimization

We now turn to constrained optimization problems which are, in general, very difficult to solve. Table 4 summarizes some of our computation results on some of the toughest problems from [9]. We give the number of variables in the initial statement (v), the number of constraints (c), the `CPU` time, and the number of splits. Note that, for a problem with n variables and m constraints, the system generates a constraint problem involving $n + m$ variables when using the Fritz-John conditions.

Benchmarks	v	c	Time	Splits
`h95`	6	16	2.59	10
`h96`	6	16	2.64	11
`h97`	6	16	103.16	230
`h98`	6	16	51.59	226
`h100`	7	18	53.71	131
`h106`	8	22	926.72	149
`h113`	10	28	4410.26	7296

Table 4: Summary of the Experimental Results on Constrained Optimization.

---

min

$$4.3x_1 + 31.8x_2 + 63.3x_3 + 15.8x_4 + 68.5x_5 + 4.7x_6$$

subject to

$$0 \leq x_1 \leq 0.31, 0 \leq x_2 \leq 0.046, 0 \leq x_3 \leq 0.068$$
$$0 \leq x_4 \leq 0.042, 0 \leq x_5 \leq 0.028, 0 \leq x_6 \leq 0.0134$$

$$17.1x_1 + 38.2x_2 + 204.2x_3 + 212.3x_4 + 623.4x_5 + 1495.5x_6 - 169x_1x_3 - 3580x_3x_5 - 3810x_4x_5 - 18500x_4x_6 - 24300x_5x_6 \geq b_1$$

$$17.9x_1 + 36.8x_2 + 113.9x_3 + 169.7x_4 + 337.8x_5 + 1385.2x_6 - 139x_1x_3 - 2450x_4x_5 - 16600x_4x_6 - 17200x_5x_6 \geq b_2$$

$$-273x_2 - 70x_4 - 819x_5 + 26000x_4x_5 \geq b_3,$$

$$159.9x_1 - 311x_2 + 587x_4 + 391x_5 + 2198x_6 - 14000x_1x_6 \geq b_4$$

Figure 5: The Constrained Optimization Problem for `h95, h96, h97, h98`.

Problems `h95, h96, h97, h98` are instantiations of the generic problem described in Figure 5 for the following values of b_i:

i	$b(95)$	$b(96)$	$b(97)$	$b(98)$
1	4.97	4.97	32.97	32.97
2	−1.88	−1.88	25.12	25.12
3	−29.08	−69.08	−29.08	−124.08
4	−78.02	−118.02	−78.02	−173.03

Problems `h100, h106` and `h113` are depicted in Figures 6, 7, and 8.

min

$$(x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to

$$-10^8 \leq x_1, x_2, x_3, x_4, x_5, x_6, x_7 \leq 10^8$$

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$$
$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$$
$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$$
$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0.$$

Figure 6: The Constrained Optimization Problem for `h100`.

min

$$x_1 + x_2 + x_3$$

subject to

$$100 \leq x_1 \leq 10000$$
$$1000 \leq x_2, x_3 \leq 10000$$
$$100 \leq x_4, x_5, x_6, x_7, x_8 \leq 400$$

$$0.0025(x_4 + x_6) \leq 1$$
$$0.0025(-x_4 + x_5 + x_7) \leq 1$$
$$0.01(-x_5 + x_8) \leq 1$$
$$100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \leq 0$$
$$x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \leq 0$$
$$x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \leq 0$$

Figure 7: The Constrained Optimization Problem for `h106`.

min

$$x_1^2 + x_2^2 + x_1 * x_2 - 14 * x_1 - 16 * x_2 + (x_3 - 10)^2 +$$
$$4 * (x_4 - 5)^2 + (x_5 - 3)^2 + 2 * (x_6 - 1)^2 + 5 * x_7^2 +$$
$$7 * (x_8 - 11)^2 + 2 * (x_9 - 10)^2 + (x_10 - 7)^2 + 45$$

subject to

$$0 \leq x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} \leq 10$$

$$105 - 4 * x_1 - 5 * x_2 + 3 * x_7 - 9 * x_8 \geq 0$$
$$-10 * x_1 + 8 * x_2 + 17 * x_7 - 2 * x_8 \geq 0$$
$$8 * x_1 - 2 * x_2 - 5 * x_9 + 2 * x_10 + 12 \geq 0$$
$$-3 * (x_1 - 2)^2 - 4 * (x_2 - 3)^2 - 2 * x_3^2 + 7 * x_4 + 120 \geq 0$$
$$-5 * x_1^2 - 8 * x_2 - (x_3 - 6)^2 + 2 * x_4 + 40 \geq 0$$
$$-0.5 * (x_1 - 8)^2 - 2 * (x_2 - 4)^2 - 3 * x_5^2 + x_6 + 30 \geq 0$$
$$-x_1^2 - 2 * (x_2 - 2)^2 + 2 * x_1 * x_2 - 14 * x_5 + 6 * x_6 \geq 0$$
$$3 * x_1 - 6 * x_2 - 12 * (x_9 - 8)^2 + 7 * x_10 \geq 0$$
$$x_1^2 + x_2^2 + x_1 * x_2 - 14 * x_1 - 16 * x_2 + (x_3 - 10)^2 +$$
$$4 * (x_4 - 5)^2 + (x_5 - 3)^2 + 2 * (x_6 - 1)^2 + 5 * x_7^2 +$$
$$7 * (x_8 - 11)^2 + 2 * (x_9 - 10)^2 + (x_10 - 7)^2 + 45 \leq 200$$

Figure 8: The Constrained Optimization Problem for `h113`.

5 Related Work

The introduction of a relational form of interval arithmetic in logic programming has been proposed by Cleary in [4]. These ideas have been developed and made popular by the CLP system BNR-Prolog [30] and generalized to constraint solving over discrete quantities in its successor CLP(BNR) [28, 2]. Many other systems (e.g [16, 36]) have been developed on similar principles. The key idea behind CLP(Intervals) languages is to let users state arbitrary constraints over reals and to narrow down the set of possible values for the variables using various approximations of arc-consistency In addition, combining the constraint solver with splitting operations allows these systems to isolate narrow regions which may contain solutions to sets of constraints. Traditionally, CLP(Intervals) languages were designed in terms of simple primitive constraints (e.g. *add(a,y,z)*, *mult(x,y,z)*, cos*(x,z)*, ...) on which they apply approximations of arc-consistency. Complex constraints are simply rewritten into a set of primitive constraints. The advantage of this methodology is the elegant operational semantics of the language. The inconvenience is that convergence may be slow and the pruning is relatively weak due to the decomposition process.

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 interval Newton methods 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 traditional Newton 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 traditional 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 interesting to note that the idea of using approximations of arc-consistency was also used independently by Hong and Stahl [10], 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.

6 Conclusion

This paper has presented Newton, a constraint programming language over nonlinear constraints. Newton uses intervals to cope with the two fundamental problems of this application area: numerical

accuracy and computational complexity. Numerical accuracy is dealt with by evaluating every operation on intervals instead of on floating-point numbers. The computational difficulty of the problem is addressed by associating intervals with variables and by using constraints to reduce these intervals. The interval reduction is performed using a novel consistency notion, box-consistency, which can be applied to various interval extensions to produce well-known and novel pruning operators. `Newton` uses box-consistency to solve constraint systems as well as unconstrained and constrained optimization problems. Experimental results on numerous benchmarks from numerical analysis, and comparison with other tools, show the effectiveness of `Newton`.

Acknowledgments

The equation-solving algorithm was developed jointly with D. McAllester and D. Kapur [37]. The formalization of box-consistency was developed jointly with F. Benhamou [1]. Special thanks to A. Colmerauer and B. Le Charlier for several helpful comments and suggestions, and to P. Codognet for his invitation to `CCP'95`. This work was supported in part by the Office of Naval Research under grant ONR Grant N00014-94-1-1153 and a NSF National Young Investigator Award with matching funds of Hewlett-Packard.

References

[1] 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.

[2] F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1994. To appear.

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

[4] J.G Cleary. Logical Arithmetic. *Future Generation Computing Systems*, 2(2):125–149, 1987.

[5] 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.

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

[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] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems. Springer Verlag, 1981.

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

[11] Moré. J., B. Garbow, and K. Hillstrom. Testing Unconstrained Optimization Software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

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

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

[14] R.B. Kearfott. A Review of Techniques in the Verified Solution of Constrained Global Optimization Problems. (To Appear), 1994.

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

[16] J.H.M. Lee and M.H. van Emden. Interval Computation as Deduction in CHIP. *Journal of Logic Programming*, 16(3-4):255–276, 1993.

[17] A.V. Levy and A. Montalvo. The Tunnelling Algorithm for the Global Minimization of Functions. *SIAM J. Sci. Stat. Comput.*, 6(1):15–29, 1985.

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

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

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

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

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

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

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

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

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

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

[28] W. Older and F. Benhamou. Programming in CLP(BNR). In *PPCP'93*, Newport, RI (USA), 1993.

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

[30] W. Older and A. Vellino. Constraint Arithmetic on Real Intervals. In *Constraint Logic Programming: Selected Research*. The MIT Press, Cambridge, Massachussetts, 1993.

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

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

[33] D. Ratz. Box-Splitting Strategies for the Interval Gauss-Seidel Step in a Global Optimization Method. *Computing*, 53(3-4):337–353, 1994.

[34] 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.

[35] H. Schwefel. *Numerical Optimization of Computer Models*. Wiley, New York, 1981.

[36] G. Sidebottom and W. havens. Hierarchical Arc Consistency Applied to Numeric Processing in Constraint Logic Programming. *Computational Intelligence*, 8(4), 1992.

[37] P. Van Hentenryck, D. McAllister, and D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM Journal on Numerical Analysis*, 1995. (to appear).

[38] 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.