ELSEVIER

Artificial Intelligence 103 (1998) 209–235

Artificial Intelligence

A gentle introduction to NUMERICA

Pascal Van Hentenryck [1]

Brown University, Box 1910, Providence, RI 02912, USA

Abstract

NUMERICA is a modeling language for stating and solving global optimization problems. It makes it possible to express these problems in a notation close to the way these problems are stated in textbooks or scientific papers. In addition, the constraint-solving algorithm of NUMERICA, which combines techniques from numerical analysis and artificial intelligence, provides many guarantees about correctness, convergence, and completeness.

This paper is a gentle introduction to NUMERICA. It highlights some of the main difficulties of global optimization and illustrates the functionality of NUMERICA by contrasting it to traditional methods. It also presents the essence of the constraint-solving algorithm of NUMERICA in a novel, high-level, way.

1. Introduction

Many science and engineering applications require the user to find solutions to systems of nonlinear constraints over real numbers or to optimize a nonlinear function subject to nonlinear constraints. This includes applications such as the modeling of chemical engineering processes and of electrical circuits, robot kinematics, chemical equilibrium problems, and design problems (e.g., nuclear reactor design). The field of global optimization is the study of methods to find all solutions to systems of nonlinear constraints and all global optima to optimization problems. Nonlinear problems raise many issues from a computation standpoint. On the one hand, deciding if a set of polynomial constraints has a solution is NP-hard. In fact, Canny [4] and Renegar [36] have shown that the problem is in PSPACE and it is not known whether the problem lies in NP. Nonlinear programming problems can be so hard that some methods are designed only to solve problems up to, say, 20 variables. On the other hand, computing over real numbers raises numerical problems because of the finite nature of computers.

[1] E-mail: pvh@cs.brown.edu.

0004-3702/98/$ – see front matter
PII: S0004-3702(98)00053-8

NUMERICA [39] is a modeling language for global optimization which makes it possible to solve nonlinear problems written in a form close to the statements traditionally found in textbooks and scientific papers. In addition, and contrary to most nonlinear programming tools, NUMERICA provides many guarantees on its results (modulo implementation errors):

- *Correctness*: NUMERICA never produces any wrong solution;
- *Completeness*: Under reasonable assumptions, NUMERICA is guaranteed to isolate all solutions to nonlinear equation systems and all global optima to unconstrained and constrained optimization problems;
- *Finiteness*: NUMERICA is guaranteed to converge;
- *Certainty*: NUMERICA can prove the existence of solutions and the absence of solutions.

These functionalities should be contrasted with traditional numerical methods (e.g., quasi-Newton methods). Traditional methods are inherently local: they converge quickly when they are close to a solution or to a local optimum but it is outside the scope of these methods to find all solutions (or global optima) or to prove the existence or absence of solutions. Traditional methods may also fail to converge on hard problems.

The limitations of local methods come from their inability to obtain *global* information on nonlinear functions. There is no way to collect global information on a function by probing finitely many points. In contrast, NUMERICA has the ability to evaluate nonlinear functions over intervals, which provides global information on the value of the function on any point in the intervals. The global nature of this information makes it possible to bound numerical errors automatically and to prune away entire regions of search space. As a consequence, the use of intervals makes it possible to implement global search algorithms for nonlinear programming.

Of course, the use of intervals in numerical computations is hardly new, since it originated from Moore's thesis in 1966 [28] and is a very active research area (e.g., [12–15, 18,20–23,28,32,37]). What distinguishes the constraint-solving algorithm of NUMERICA is the combination of techniques from numerical analysis and artificial intelligence to obtain effective pruning techniques (for many problems). At a very abstract level, NUMERICA can be viewed as mapping continuous problems into discrete problems, which is exactly the opposite of traditional relaxation techniques (e.g., in integer programming [10]). Once nonlinear programming problems are viewed as discrete problems, it is natural to apply consistency techniques such as arc- and path-consistency (e.g., [25–27]) which have been successfully applied in many areas [40].

NUMERICA, and its constraint-solving algorithm, does not aim at replacing local methods. Local methods are extremely effective tools when they apply and are probably the only way to approach large-scale nonlinear programming problems involving thousands of variables. However, there are many applications where the additional functionalities of NUMERICA are needed, either because of the nature of the application, or because the problem is too hard for local methods, or simply because the robustness of the approach simplifies the task. This is especially true for small-scale highly nonlinear problems as those found in chemical and electrical engineering where traditional methods are likely to diverge, are unable to locate all solutions or to prove the absence of solutions (a requirement in these problems). Gehrke and Marquardt [11] in fact indicate that progress in chemical engineering increases the need for these functionalities.

The rest of this paper is organized as follows. Section 2 discusses the nature of nonlinear programming problems, including theoretical limitations and practical difficulties. Section 3 is a short tour of NUMERICA, which depicts its functionality on a variety of problems and contrasts it to traditional methods. Section 4 is a novel, very high-level, presentation of the main constraint-solving algorithm used in NUMERICA. Section 5 concludes the paper and suggests directions for further research. More information about NUMERICA can be found in [38,39].

2. The nature of nonlinear programming

This section discusses some of the properties of nonlinear systems and some of the limitations of traditional methods.

2.1. *What is possible and what is not?*

Today's computers can manipulate and store only a finite amount of information. Since the solution of a nonlinear problem may be a real number that cannot be represented in finite space or displayed on a screen in finite time, the best we can hope for in general is a point close to a solution (preferably with some guarantee on its proximity to the solution) or an interval enclosing a solution.

Computer methods for solving nonlinear problems typically use floating-point numbers to approximate real numbers. Since there are only finitely many floating-point numbers, these methods are bound to make numerical errors. These errors, although probably small considered in isolation, may have fundamental implications on the results. Consider, for instance, Wilkinson's problem, which consists in finding all solutions to the equation

$$\prod_{i=1}^{20}(x+i)+px^{19}=0$$

in the interval $[-20.4,-9.4]$. When $p=0$, the equation obviously has 11 solutions. When $p=2^{-23}$, it has no solution. Wilkinson's problem clearly indicates that a small numerical error (e.g., assume that p is the output of some numerical computation) can have fundamental implications for the results of an application. These numerical issues require users of numerical software to exercise great care when interpreting their results. With this in mind, consider the following combustion problem, which consists of finding positive values for x_i $(1 \leqslant i \leqslant 10)$ satisfying the equations

$$\begin{aligned}
&x_2+2x_6+x_9+2x_{10}=10^{-5},\\
&x_3+x_8=3\cdot 10^{-5},\\
&x_1+x_3+2x_5+2x_8+x_9+x_{10}=5\cdot 10^{-5},\\
&x_4+2x_7=10^{-5},\\
&0.5140437\,10^{-7}\,x_5=x_1^2,\\
&0.1006932\,10^{-6}\,x_6=2x_2^2,
\end{aligned}$$

$$0.7816278\,10^{-15}\,x_7 = x_4^2,$$
$$0.1496236\,10^{-6}\,x_8 = x_1x_3,$$
$$0.6194411\,10^{-7}\,x_9 = x_1x_2,$$
$$0.2089296\,10^{-14}\,x_{10} = x_1x_2^2.$$

Using $(0.5,\ldots,0.5)$ as starting point and the default setting of the system, a well-known commercial system produces a point, say a. In the same conditions but with the defaults set to obtain the highest numerical precision, the same commercial system produces another point, say b, and prints a warning that the machine precision is not sufficient to achieve the desired accuracy. It is not obvious in this case how to interpret these results in a meaningful way.

It is also interesting to mention the common belief that proving the existence or uniqueness of solutions is outside the scope of computer algorithms. For instance, Dennis and Schnabel in their excellent text [8] present the three functions

$$f_1(x) = x^4 - 12x^3 + 47x^2 - 60x,$$
$$f_2(x) = x^4 - 12x^3 + 47x^2 - 60x + 24,$$
$$f_3(x) = x^4 - 12x^3 + 47x^2 - 60x + 24.1$$

and state:

> *It would be wonderful if we had a general-purpose computer routine that would tell us: "The roots of $f_1(x)$ are $x = 0, 3, 4$, and 5; the real roots of $f_2(x)$ are $x = 1$ and $x \cong 0.888$; $f_3(x)$ has no real roots."*
> *It is unlikely that there will ever be such a routine. In general, the questions of existence and uniqueness—does a problem have a solution and is it unique?—are beyond the capabilities one can expect of algorithms that solve nonlinear problems. In fact, we must readily admit that for any computer algorithm there exist nonlinear functions (infinitely continuously differentiable, if you wish) perverse enough to defeat the algorithm. Therefore, all a user can be guaranteed from any algorithm applied to a nonlinear problem is the answer, "An approximate solution to the problem is ..." or "No approximate solution to the problem was found in the allocated time."*

This statement is not correct in general and applies mainly to local methods. Such wonderful procedures in fact exist (within the limits imposed by the finite nature of computers) and one of them is used in NUMERICA.

2.2. *Local versus global optimum*

Traditional globally convergent methods when applied to a minimization problem converge to a local optimum from almost all starting points. They are unable however to isolate all local optima and the global optima. This limitation is well illustrated by the minimization of the function

$$\sum_{i=1}^{5} i\cos\big((i+1)x + i\big)$$

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

Fig. 1. The transistor modeling problem.

and the maximization of the function

$$-\left(\sum_{i=1} 5i\cos\big((i-1)x_1 + i\big)\right)\left(\sum_{i=1} 5i\cos\big((i+1)x_2 + i\big)\right)$$

as well as by the minimization of the function $f(x_1, \ldots, x_n)$ defined as

$$10\sin(\pi y_1)^2 + (y_n - 1)^2 + \sum_{i=1}^{n-1}(y_i - 1)^2\big(1 + 10\sin(\pi y_{i+1})^2\big).$$

These functions have many local minima. For instance, the last function has 10^{10} local minima when $n = 10$ but only a single global minimum. It is unlikely that a local method will converge towards a global minimum without external knowledge about these problems.[2] Also, a local method will never be able to prove that the global minimum has been found.

Of course, finding global optima is much harder than finding local optima, since the whole search space must be explored (at least implicitly). As a consequence, there are limits to the size of problems which can be solved globally in practice. For this reason, NUMERICA also offers the possibility of finding local optima, sacrificing the completeness of the search but preserving the numerical correctness and the ability to prove existence of local optima.

2.3. Convergence

In addition to the above theoretical limitations, local methods also suffer from practical problems when implemented on a computer. One of the main problems is of course convergence. An interesting example in this context is the transistor modeling example of Ebers and Moll [9]. The problem is to find a solution to the system of nonlinear equations depicted in Fig. 1 where the variables x_i must take their values in [0, 10] and the constants are given by

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

[2] Of course, there always exists a starting point that will converge towards the global optimum.

Ratschek and Rokne [35] summarize various attempts to find a solution to this problem using local methods and states

> *In 1974, Cutteridge* [7] *combined local damped Newton–Raphson steps with the conjugate gradient method and a second-order gradient-descent method with eigenvalue determination where the two latter methods were applied to the least squares problem* [...] *Cutteridge emphasized that only the sophisticated combination of the three methods had led to a positive result, i.e., it did not suffice to only use the first two approaches mentioned above* [...].

Another important practical problem is convergence to an undesired solution, i.e., a solution that fails to satisfy some external constraints not included in the problem statement. Globally convergent algorithms are guaranteed to converge to some solution or some local minimum from almost any starting point, but they may fail to produce a given solution. For instance, a traditional quasi-Newton method applied to the transistor modeling problem almost always converges to a solution in which some variables have negative values. Solution a produced by the commercial system on the combustion problem has some negative components. Morgan [31] also mentions that these undesired convergences are typical of chemical equilibrium systems:

> *The other day an electrochemist friend came by my office with a problem. He was trying to work out part of a battery-plate manufacturing process. He had set up a math model to determine the amounts of various metal compounds that would be present in the plating bath at various times. He had ended up with a system of* 10 *polynomial equations in* 10 *unknowns. His problem was that Newton's method kept converging to nonphysical solutions.* [...] *This incident has been repeated in various guises many times.*

2.4. Practicality

The functionalities of NUMERICA of course come at a price. The intractable nature of nonlinear programming precludes any guarantee on the computation times of interval methods. Conventional wisdom claims that interval methods are too slow to be of practical use and that their guarantees and ease of use come at too high a price. The performance of NUMERICA indicates that, for a rich collection of nonlinear problems, the price to pay is reasonable. Moreover, even when the full functionality of global methods is not needed, NUMERICA avoids the tedious work necessary to tune local methods and find suitable starting points. As a consequence, NUMERICA's ease of use and robustness frequently compensate for a longer running time and may even reduce the actual time to obtain a solution. In this context, it may be useful to mention that NUMERICA takes essentially linear time in the number of variables to isolate the zeros of the Broyden banded function, a traditional benchmark from numerical analysis, even when the initial range of the variable is as large as or larger than, say, $[-10^8, 10^8]$.

In addition, NUMERICA compares well and frequently outperforms continuation methods on their benchmarks. This good performance comes from a novel combination of interval analysis methods (e.g., Hansen–Sengupta's operator) and constraint satisfaction techniques. The combination of these orthogonal techniques gives surprisingly good results

```
Input:
   real p: "p: ";
Variable:
   x in [-20.4,-9.4];
Body:
   solve system all
      prod(i in [1..20]) (x + i) + p * x^19 = 0;
```

Fig. 2. The Wilkinson problem in NUMERICA.

on many problems, although understanding its strengths and limitations more formally requires further research.

Of course, there are also classes of problems for which interval methods are not appropriate at this point because interval evaluations may lose too much precision. For instance, nonlinear least-squares problems are not amenable to effective solution with the interval methods of which we are aware. Interval methods converge, of course, on these applications but they do not compare well in efficiency with local methods.

3. A tour of NUMERICA

We now illustrate NUMERICA on a number of applications to highlight the language and its functionality.

The Wilkinson problem. Let us start with the Wilkinson's problem. The NUMERICA statement for this problem is depicted in Fig. 2. The statement declares an input constant `p` of type real and a variable `x` which ranges in `[-20.4,-9.4]`. The body of the statement requests all zeros to the Wilkinson's function. Note the aggregation operator `prod` which makes it possible to have a statement close to a LaTeX description. When `p = 0`, NUMERICA returns eleven solutions, ten of which being represented exactly. These exact solutions are of the form

```
Solution: 1 [SAFE]
  x = -20
```

The approximate solution

```
Solution: 5 [SAFE]
  x = -16.0 + [-0.356e-14 , +0.178e-14]
```

illustrates that NUMERICA returns intervals enclosing solutions. The keyword `SAFE` indicates that NUMERICA has proven that there exists a solution inside the interval. When the keyword is not present, it simply means that NUMERICA was not able to obtain a proof of existence. As a consequence, an interval (or a box when the dimension is greater than 1) not marked with `SAFE` may or may not contain a solution. When `p` $= 2^{-23}$, NUMERICA simply returns that there is no solution to the problem.

```
Pragma:
   precision = 1e-12;
Variable:
   x:array[1..10] in [0..1];
Body:
   solve system all
      x[2] + 2 * x[6] + x[9] + 2*x[10] = 1e-5;
      x[3] + x[8] = 3e-5 ;
      x[1] + x[3] + 2*x[5] + 2*x[8] + x[9] + x[10] = 5e-5;
      x[4] + 2 * x[7] = 1e-5;
      0.5140437e-7 * x[5] = x[1]^2;
      0.1006932e-6 * x[6] = 2 * x[2]^2;
      0.7816278e-15 * x[7] = x[4]^2;
      0.1496236e-6 * x[8] = x[1]*x[3];
      0.6194411e-7 * x[9] = x[1]*x[2];
      0.2089296e-14 * x[10] = x[1]*x[2]^2;
```

Fig. 3. The combustion problem in NUMERICA.

The combustion problem. Reconsider the combustion problem discussed earlier. The NUMERICA statement for this problem is shown in Fig. 3. The statement uses an array of variables and a pragma to specify the desired accuracy. NUMERICA returns the unique positive solution

```
Solution: 1 [SAFE]
  x[1] = 0.00000014709 + [-0.3151616090976e-12 ,+0.5348538119178e-12]
  x[2] = 0.00000022619636102 + [0.31239815e-17 , 0.67358889e-17]
  x[3] = 0.000015128076338 + [0.28151603e-15 , 0.38648037e-15]
  x[4] = 0.000000000062
         + [0.39268906099084274e-12 , 0.51871986419385418e-12]
  x[5] = 0.0000004208884800 + [0.670113679e-16 , 0.922409855e-16]
  x[6] = 0.000001016251221 + [0.301906781e-15 , 0.402067581e-15]
  x[7] = 0.000004999968 + [0.740640067315e-12 , 0.803655470521e-12]
  x[8] = 0.000014871923661 + [0.60945464e-15 , 0.7225471e-15]
  x[9] = 0.0000005371172945 + [0.142273089e-16 , 0.563773625e-16]
  x[10] = 0.000003602091950 + [0.808352357e-15 , 0.927440531e-15]
```

and proves its existence in about 0.1 second on a Sun Sparc-10. Solution b produced by the commercial system mentioned previously is close to being contained in this output box.

Dennis and Schnabel's functions. Let us now revisit the Dennis and Schnabel's functions

$$f_1(x) = x^4 - 12x^3 + 47x^2 - 60x$$

$$f_2(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$$

$$f_3(x) = x^4 - 12x^3 + 47x^2 - 60x + 24.1.$$

NUMERICA returns four boxes enclosing the solutions and proves the existence of a solution in each of them for f_1; it returns two boxes and proves the existence of a solution in each of them for f_2; it shows the absence of solutions for f_3. The computation times for

these examples are negligible. More precisely, the NUMERICA statement

```
Variable:
   x in [0..1e8];
Body:
   solve system all
      x^4 - 12 * x^3 + 47 * x^2 - 60 * x + 24 = 0;
```

produces the following output boxes:

```
Solution: 1 [SAFE]
  x = 0.8883057790717 + [0.4e-13 , 0.6e-13]

Solution: 2 [SAFE]
  x = 1.0 + [-0.1e-13 , +0.1e-13]
```

This example illustrates well the functionalities of NUMERICA compared to traditional methods.

The Broyden banded function. We now consider a traditional benchmark from numerical analysis: the Broyden Banded function. This is a benchmark which is found in traditional collections of problems (e.g., [19]) and which is interesting to illustrate several aspects of NUMERICA. The problem amounts to finding the zeros of n functions

$$f_i(x_1,\ldots,x_n) = x_i\left(2+5x_i^2\right)+1-\sum_{k\in J_i} x_k(1+x_k)\ (1\leqslant i\leqslant n)$$

where $J_i = \{j \mid \max(1,i-5)\leqslant j\leqslant \min(n,i+1)\ ,\ j\neq i\}$. Note that this statement uses n sets that share the same basic definition.

Fig. 4 depicts the NUMERICA statement and it contains several interesting features. First, it illustrates the use of generic sets in the instructions

```
Set:
   J[i in idx]
      = { j in [max(1,i-5)..min(n,i+1)] | j <> i };
```

to obtain a close similarity with the mathematical statement. Second, it also illustrates the use of generic constraints

```
[i in idx]:
   0 = x[i] * (2 + 5 * x[i]^2) + 1
      - Sum(k in J[i]) x[k] * (1 +x[k]);
```

to state a system of equations. Both of these features simplify the statement significantly.

Also interesting is the performance behavior of NUMERICA as shown in Table 1. Experimentally, it was observed that NUMERICA essentially encloses the unique solution

```
Input:
   int n : "Number of variables: ";
Constant:
   range idx = [1..n];
Set:
   J[i in idx] = { j in [max(1,i-5)..min(n,i+1)] | j <> i };
Variable:
   x : array[idx] in [-10e8..10e8];
Body:
   solve system all
     [i in idx]:
        0 = x[i] * (2 + 5 * x[i]^2) + 1 - Sum(k in J[i]) x[k] * (1 +x[k]);
```

Fig. 4. The Broyden banded function.

Table 1
Performance results on Broyden's function

n	Solving time (ms)	Growth factor
5	100	
10	500	5.00
20	1600	3.20
40	4200	2.65
80	9800	2.33
160	30700	3.13

in linear time in the number of variables for extremely large range.[3] This is not an isolated case and there are many benchmarks which exhibit a similar behavior. Understanding why and isolating this class of problems is an interesting and open theoretical problem.

The transistor modeling problem. As our last example of equation solving, reconsider the transistor problem. The NUMERICA statement is shown in Statement 5. Note the pragmas which specifies the level of constraint propagation (i.e., 2) and the search strategy (i.e., splitting the largest interval first): these will be explained in the next section. NUMERICA finds the unique solution to the transistor modeling problem in the box $[0\ldots10]^9$ and proves its existence and the absence of other solutions in less than 40 minutes. Traditional commercial tools are unable to locate the solution to this problem. The previous interval solution [35] required more than 14 months on a network of Sun-1 workstations.

Unconstrained minimization. Statement 6 depicts the NUMERICA statement for one of the unconstrained minimization problems discussed previously. The statement illustrates the use of functions (as abbreviations of complex expressions), of trigonometric functions, and of real constants such as `pi`. On optimization problems, NUMERICA is guaranteed to

[3] There is a cubic step at the end of the search to prove the existence of a solution.

```
Pragma:
   variable_choice = lf;
   consistency = 2;

Constant:
   range xr = [1..5];
   range yr = [1..4];
   real g: array[xr,yr] = [
      [ 0.485, 0.752, 0.869, 0.982 ],
      [ 0.369, 1.254, 0.703, 1.455 ],
      [ 5.2095, 10.0677, 22.9274, 20.2153 ],
      [ 23.3037, 101.779, 111.461, 191.267 ],
      [ 28.5132, 111.8467, 134.3884, 211.4823 ]];

Variable:
   x: array[1..9] in [0.0..10.0];
   y: array[0..2] in [-1000..1000];

Body: solve system all
   y[0] = 1 - x[1] * x[2];
   y[1] = y[0] * x[3];
   y[2] = y[0] * x[4];
   [k in yr]:
      y[1] * (exp(x[5] * (g[1,k] - g[3,k]*x[7]*1e-3-g[5,k]*x[8]*1e-3)) - 1)
      - g[5,k] + g[4,k]*x[2] = 0;
   [k in yr]:
      y[2] * (exp(x[6]*(g[1,k]-g[2,k]-g[3,k]*x[7]*1e-3
      +g[4,k]*x[9]*1e-3)) -1) - g[5,k]*x[1] + g[4,k] = 0;
   x[1]*x[3] - x[2]*x[4] = 0;
```

Fig. 5. The transistor modeling problem in NUMERICA.

bound and isolates all global optima. Table 2 gives the performance results on this problem. As can be seen, NUMERICA seems to be essentially quadratic in the number of variables on this problem.

Constrained optimization. We conclude this section by showing a statement for solving a constrained optimization problem in NUMERICA. Statement 7 in fact depicts problem 95 from a standard collection of benchmarks [17]. Once again, NUMERICA is guaranteed to isolate global optima in constrained optimization problems.

4. The essence of NUMERICA

The purpose of this section is to present the main ideas behind NUMERICA's implementation. The presentation here is novel and aims at crystallizing the main intuitions behind the algorithm. It starts with a review of the main concepts of interval analysis and describes the problem to be solved, the main algorithm, and the pruning techniques. The main algorithm is then reconsidered to remove some of the simplifying assumptions. Throughout this section, only equation solving is considered, since it is also the cornerstone

```
Input:
   int n : "Number of variables";
Constant:
   range idx = [1..n];
Variable:
   x : array[idx] in [-10..10];
Function:
   y(i in idx) = 1 + 0.25 * (x[i]-1);
Body:
   minimize
      10 * sin(pi*y(1))^2 + (y(n) - 1)^2
      + Sum(i in [1..n-1])
          (y(i) -1)^2 * (1 + 10 * sin(pi*y(i+1))^2);
```

Fig. 6. Unconstrained optimization: Levy 8'.

Table 2
Performance results on Levy 8'

n	Solving time (s)	Growth factor
5	0.40	
10	1.20	3.00
20	4.30	3.58
40	27.10	6.30
80	136.60	5.04

for optimization problems. Indeed, optimality conditions for optimization problems (e.g., the Kuhn–Tucker conditions) can be expressed as a system of equations.

4.1. Preliminaries

Here we review some basic concepts of interval analysis to needed for this paper. More information on interval arithmetic can be found in many places (e.g., [1,14,15,28,29,32, 34]). Our definitions are slightly non-standard.

4.1.1. Interval arithmetic

We consider $\mathbb{R}^{\infty} = \mathbb{R} \cup \{-\infty, \infty\}$, the set of real numbers extended with the two infinity symbols, and the extension of the relation $<$ to this set. We also consider a finite subset $\mathcal{F}$ of $\mathbb{R}^{\infty}$ 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

$$\{r \in \mathbb{R} \mid l \leqslant r \leqslant u\}.$$

The set of intervals is denoted by $\mathcal{I}$ and is ordered by set inclusion.[4]

[4] These intervals are usually called floating-point intervals in the literature.

```
Constant:
  real B1 = 4.97;
  real B2 = -1.88;
  real B3 = -29.08;
  real B4 = -78.02;

Variable:
  x: array[1..6];

Body:
  minimize
    4.3 * x[1] + 31.8 * x[2] + 63.3 * x[3] + 15.8 * x[4] +
    68.5 * x[5] + 4.7 * x[6]
  subject to
    [i in [1..6]]: x[i] >= 0;
    x[1] <= 0.31; x[2] <= 0.046; x[3] <= 0.068;
    x[4] <= 0.042; x[5] <= 0.028; x[6] <= 0.0134;

    17.1 * x[1] + 38.2 * x[2] + 204.2 * x[3] + 212.3 * x[4]
    + 623.4 * x[5] +
    1495.5 * x[6] - 169 * x[1] * x[3] - 3580 * x[3] * x[5]
    - 3810 * x[4] * x[5] - 18500 * x[4] * x[6] - 24300 * x[5] * x[6] >= B1;

    17.9 * x[1] + 36.8 * x[2] + 113.9 * x[3] + 169.7 * x[4]
    + 337.8 * x[5] + 1385.2 * x[6] - 139 * x[1] * x[3] - 2450 * x[4] * x[5]
    - 16600 * x[4] * x[6] - 17200 * x[5] * x[6] >= B2;

    -273 * x[2] - 70 * x[4] - 819 * x[5] + 26000 * x[4] * x[5] >= B3;

    159.9 * x[1] - 311 * x[2] + 587 * x[4] + 391 * x[5]
    + 2198 * x[6] - 14000 * x[1] * x[6] >= B4;
```

Fig. 7. A constrained optimization problem in Numerica.

Definition 2 (*Enclosure*). Let S be a subset of $\mathbb{R}$. 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 $r \in \mathbb{R}$.

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^+ (respectively l^-) to denote the smallest (respectively largest) $\mathcal{F}$-number strictly greater (respectively smaller) than the $\mathcal{F}$-number l. To capture outward rounding, we use $\lceil r \rceil$ (respectively $\lfloor r \rfloor$) to return the smallest (respectively largest) $\mathcal{F}$-number greater (respectively 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 $\langle r_1, \ldots, r_n \rangle$. $\mathbb{Q}$ is the set of rational numbers and $\mathbb{N}$ is the set of natural numbers. We also use the following notations:

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

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

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

Definition 4 (*Set extensions*). Consider a set $S \subseteq \mathbb{R}^n$ and a function $f : \mathbb{R}^n \to \mathbb{R}$. The set extension of f is defined as

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

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

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

Example 6. The interval function $\oplus$ defined as

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

is an interval extension of addition of real numbers.

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

Definition 7 (*Monotonic interval extensions*). An interval function $F : \mathcal{I}^n \to \mathcal{I}$ is *monotonic* in $\vec{I}_0$ if

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

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

4.1.2. Constraint representations

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

4.2. Problem description

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

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

in a box $\vec{I}^0 = \langle I_1^0, \ldots, I_n^0 \rangle$. Of course, on a computer, it is generally impossible to find these solutions exactly and interval methods aim at returning small boxes containing all solutions. Preferably, each such box is *safe*, meaning that it contains a solution. For the purposes of this section, interval methods can thus be viewed as solving the following problem: assuming that F_i is a monotonic interval extension of f_i $(1 \leqslant i \leqslant n)$, find all canonical[5] boxes $\langle I_1, \ldots, I_n \rangle$ in $\vec{I}^0$ satisfying

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

This is of course a simplification, since interval methods generally use various interval extensions. However, restricting attention to this problem has the benefits of crystallizing the intuition underlying our novel pruning methods. Section 4.7 reconsiders this simplification.

Notation. Let $\mathcal{S}$ be a system of constraints of the form

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

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

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

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

4.3. The generic branch-and-prune algorithm

The above problem description highlights the finite nature of the problem, since there are only finitely many floating-point intervals. Most interval methods are thus best viewed

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

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

Fig. 8. The branch-and-prune algorithm.

as global search algorithms iterating two main steps: pruning and branching. The basic schema underlying these algorithms, the branch-and-prune schema, is depicted in Fig. 8. The function `Search` receives a system of interval constraints $\mathcal{S}$ and an initial box $\vec{I}^0$: it returns the set of canonical boxes $\vec{I}$ in $\vec{I}^0$ satisfying $\mathcal{S}(\vec{I})$. The function `Search` first applies a pruning step that reduces the initial box. This pruning step is the main topic of this section. If the resulting box $\vec{I}$ is empty, there is no solution to the problem. If the box $\vec{I}$ is canonical, it is returned as a result. Otherwise, the box is split along one dimension into two subboxes, $\vec{I}_1$ and $\vec{I}_2$, which are then explored recursively using the same algorithm.

A specific interval algorithm can be obtained by specifying the splitting strategy and pruning techniques. Our algorithms use a strategy that consists of splitting the intervals associated with the variables in a round-robin strategy.[6] The main novelty of our algorithms lies in the pruning techniques and we will define three pruning operators, $\texttt{Prune}_0$, $\texttt{Prune}_1$, and $\texttt{Prune}_2$, that produce three distinct algorithms. These last two algorithms are used in NUMERICA.

4.4. *Box*(0)-*consistency*

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

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

[6] This default can be overwritten with pragmas: for instance, the transistor modeling statement specifies that the variable with the largest interval should be split first.

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

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

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

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

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

4.5. *Box*(1)-*consistency*

This section presents the first pruning operator used in our algorithm. It starts with an informal discussion, then specifies the pruning operator, and presents a simple implementation. [7]

4.5.1. *Informal presentation*

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

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

is thus approximated by

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

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

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

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

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

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

[7] Separating the specification from the implementation has the advantage of distinguishing what is being computed from how to compute it. There are many ways to implement the concepts described in this section, and our goal here is to focus on the concepts, not on the technical details (which can be found elsewhere [38]).

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

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

and let I_l be the leftmost interval in I_1 satisfying

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

and I_r the rightmost interval in I_1 satisfying

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

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

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

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

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

4.5.2. The pruning operator

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

Definition 8 (*Box(1)-consistency*). Let $\mathcal{S}$ be a system of interval constraints, let $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$, and let $l_i = \mathit{left}(I_i)$ and $u_i = \mathit{right}(I_i)$ $(1 \leqslant i \leqslant n)$. $\mathcal{S}$ is box(1)-consistent in $\vec{I}$ with respect to i if

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

and

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

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

The pruning operator associated with box(1)-consistency simply returns the largest box in the initial interval that is box(1)-consistent (or, more informally, the largest box in the

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

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

Fig. 9. Implementing box(1)-consistency.

initial interval that cannot be narrowed further). This box always exists because of the monotonicity of interval extensions and is unique. Of course, it can be empty.

Definition 9 (*Box*(1)*-consistency pruning*). Let $\mathcal{S}$ be a system of interval constraints and let $\vec{I}$ be a box. The pruning operator associated with box(1)-consistency can be defined as

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

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

4.5.3. A simple implementation

The pruning operator can be computed in many ways. Fig. 9 presents a simple iterative algorithm for this purpose; see [38] for a more efficient implementation. The algorithm is a simple fixpoint algorithm that terminates when no further pruning can be obtained (i.e., $\vec{I} = \vec{I}_p$). The body of the loop applies a narrowing operation on each of the variables and produces a new box that is the intersection of all these narrowings. The narrowing function $\mathtt{Narrow}_{\mathrm{I}}(\mathcal{S}, \vec{I}, i)$ returns the largest box $\vec{I}'$ in $\vec{I}$ such that $\mathcal{S}$ is box(1)-consistent in $\vec{I}'$ with respect to i.

4.6. Box(2)*-consistency*

Box consistency has been shown to be effective for solving a variety of nonlinear applications [38]. For some of the more difficult applications (e.g., the transistor modeling problem and chemical engineering applications [11]), however, better performance can be obtained by using a stronger local consistency condition that we call *box*(2)*-consistency*. Box(2)-consistency is in fact an approximation of path consistency [27] and is related to some consistency notions presented in [24].

4.6.1. Informal presentation

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

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

is thus approximated by

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

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

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

can be tested by considering all pairs of canonical intervals in I_1 and I_2 for X_1 and X_2.

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

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

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

4.6.2. The pruning operator

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

Definition 10 (*Box*(1)-*satisfiability*). A system of interval constraints $\mathcal{S}$ is *box*(1)-*satisfiable* in $\vec{I}_0$, denoted by $BoxSat_1(\mathcal{S}, \vec{I}_0)$, if there exists a box $\vec{I} \subseteq \vec{I}_0$ such that $\mathcal{S}$ is box(1)-consistent in $\vec{I}$.

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

Definition 11 (*Box*(2)-*consistency*). Let $\mathcal{S}$ be a system of interval constraints and $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$ with $I_j = [l_j, u_j]$. $\mathcal{S}$ is *box*(2)-*consistent* in $\vec{I}$ with respect to i if

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

when $l_i \neq u_i$ and if

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

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

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

Proposition 12. *Let $\mathcal{S}$ be a system of monotonic interval constraints. If $\mathcal{S}$ is box*(2)*-consistent in $\vec{I}$, then $\mathcal{S}$ is box*(1)*-consistent in $\vec{I}$.*

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

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

or, more explicitly,

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

It follows that

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

and, by monotonicity of F_i, that

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

The proof that

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

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

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

Definition 13 (*Box*(2)*-consistency pruning*). Let $\mathcal{S}$ be a system of interval constraints and let $\vec{I}$ be a box. The pruning operator associated with box(2)-consistency can be defined as

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

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

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

function Narrow2(S, ⟨I1, ..., In⟩, i)
begin
    C := {Ic ⊆ Ii | Ic is canonical and ¬ Empty(Prune1(S, ⟨I1, ..., Ii−1, Ic, Ii+1, ..., In⟩))};
    if C = ∅ then
        return ∅;
    else
        return [ min_{I∈C} left(I), max_{I∈C} right(I) ];
end
```

Fig. 10. Implementing box(2)-consistency.

4.6.3. A simple implementation

Once again, the pruning operator can be computed in many ways. Fig. 10 presents a simple iterative algorithm close to our actual implementation. The algorithm is again a simple fixpoint algorithm that terminates when no further pruning can be obtained. The body of the loop applies a narrowing operation on each of the variables and produces a new box that is the intersection of all these narrowings. The narrowing function $\mathtt{Narrow}_2(\mathcal{S}, \vec{I}, i)$ returns the largest box $\vec{I}'$ in $\vec{I}$ such that $\mathcal{S}$ is box(2)-consistent in $\vec{I}'$ with respect to i. The pruning operator of box(1)-consistency is used to compute this narrowing operator. Note that it would be easy to define any level of box-consistency at this point, since box$(k-1)$-consistency can be used to define box(k)-consistency in a generic way.

4.6.4. Related work

It is useful to relate these pruning operators to earlier work in constraint satisfaction. Projections, and approximations of projections, have been used extensively in the artificial intelligence community (under the name *consistency techniques*) to solve discrete combinatorial problems (e.g., [25,27]). They have been adapted to continuous problems (e.g., [6,24]) and used inside systems such as BNR-PROLOG and CLP(BNR) [3,33] and many systems since then. The techniques used in systems like BNR-PROLOG and CLP(BNR) are weaker than box(1)-consistency, since they decompose all constraints into ternary constraints on distinct variables before applying a form of box(1)-consistency. They do not scale well on difficult problems. Box(1)-consistency was introduced in [2]. It is related to the techniques presented in [18], which uses a similar idea for the splitting process. The consistency notions of [24] are a weaker, and less effective, form of box(k)-consistency: it is obtained by decomposing all constraints into ternary constraints over distinct variables and by applying a form of box(k)-consistency on the resulting constraints.

4.7. The branch-and-prune algorithm revisited

We now reconsider the assumptions of Section 4.2. As mentioned in that section, our algorithm uses two interval extensions, the natural interval extension and the mean value interval extension, since distinct interval extensions may produce different prunings. In particular, the natural interval extension is generally better when far from a solution, while the mean value interval extension is more precise when close to a solution. This section reviews both extensions and reconsiders the overall branch-and-prune algorithm.

4.7.1. The natural interval extension

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

Example 14 (Natural interval extension). The natural interval extension of the function $x_1(x_2 + x_3)$ is the interval function $X_1(X_2 + X_3)$.

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

Definition 15 (*Natural interval extension of a system*). Let S be a system of constraints

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

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

$$\widehat{S} = \begin{cases} 0 = \widehat{f}_1(X_1, \ldots, X_n) \\ \ldots \\ 0 = \widehat{f}_n(X_1, \ldots, X_n). \end{cases}$$

The following result is easy to prove by induction.

Proposition 16. *The natural interval extension of a function, a constraint, or a system of constraints is monotonic.*

4.7.2. The mean value interval extension

The second interval extension is based on the Taylor expansion around a point. This extension is an example of centered forms that are interval extensions introduced by Moore [28] and have been studied by many authors, since they have important properties.

The mean value interval extension of a function is parametrized by the intervals for the variables in the function. It also assumes that the function has continuous partial derivatives. Given these assumptions, 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 17 (*Mean value interval extension*). Let $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$ and m_i be the center of I_i. The mean value interval extension of a function f in $\vec{I}$, denoted by $\tau(f, \vec{I})$, is the interval function

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

Let S be a system of constraints

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

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

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

Note that the mean value interval extensions is defined in terms of natural interval extensions. The proof of the following proposition can be found in [5].

Proposition 18. *The mean value interval extension of a function is a monotonic interval extension.*

It is interesting to note that box consistency on the mean value interval extension of a system of constraints is closely related to the Hansen–Sengupta operator [15], which is an improvement over Krawczyk's operator [23]. Hansen and Smith also argue that these operators are more effective when the interval Jacobian of the system is diagonally dominant [16] and they suggest conditioning the system S. For the purpose of this section, we simply assume that we have a conditioning operator $cond(S, \vec{I})$ and use the notation $\tau_c(S, \vec{I})$ to denote $\tau(cond(S, \vec{I}), \vec{I})$. See [20,21] for extensive coverage of conditioners.

4.7.3. The branch-and-prune algorithm

We are now in position to reconsider our branch-and-prune algorithm. The new version, given in Fig. 11, differs in two ways from the algorithm presented in Section 4.3. On the one hand, the algorithm receives as input a system of constraints (instead of a system of

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

Fig. 11. The branch-and-prune algorithm revisited.

interval constraints). On the other hand, the operation `Prune` receives a system of interval constraints consisting of the natural interval extension and the conditioned mean value interval extensions of the original system.

4.7.4. Existence proof

We now describe how the algorithm proves the existence of a solution in a box. Let $\{f_1 = 0, \ldots, f_n = 0\}$ be a system of equations over variables $\{x_1, \ldots, x_n\}$, let $\vec{I} = \langle I_1, \ldots, I_n \rangle$ be a box and define the intervals I_i' $(1 \leqslant i \leqslant n)$ as follows

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

where $m_i = center(I_i)$. If

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

then there exists a zero in $\langle I_1', \ldots, I_n' \rangle$. A proof of this result can be found in [30].

5. Conclusion, challenges, and opportunities

This paper gave a gentle introduction to NUMERICA, a modeling language for solving global optimization problems. The functionalities of NUMERICA have been illustrated and contrasted with traditional methods. The essence of the constraint-solving algorithm of NUMERICA was presented at a very-high level.

There are many possible ways to improve global methods for nonlinear programming and we mention some of them without trying to be exhaustive. A particularly interesting research avenue (studied by F. Benhamou and D. Kapur for instance) is the combination of symbolic and numerical methods. New pruning techniques with a more global view of the problem is also of paramount importance to improve the pruning when far from a solution. Similarly, it would be interesting to study ways of collecting global information beyond intervals. Finally, constraint satisfaction techniques have been a driving force behind the development of NUMERICA but only a tiny fraction of the existing research is exploited in NUMERICA. It is an exciting field and it is likely to evolve substantially in the coming years.

Acknowledgement

NUMERICA was developed jointly with Laurent Michel and Yves Deville. NUMERICA is also based on previous work on `Newton` with Frédéric Benhamou, Deepak Kapur, and David McAllester. Thanks to all of them for their invaluable contributions. This work was supported by an NSF National Young Investigator Award.

References

[1] G. Alefeld, J. Herzberger, Introduction to Interval Computations, Academic Press, New York, 1983.
[2] F. Benhamou, D. McAllester, P. Van Hentenryck, CLP(intervals) revisited, in: Proc. International Symposium on Logic Programming (ILPS-94), Ithaca, NY, November 1994, pp. 124–138.
[3] F. Benhamou, W. Older, Applying interval arithmetic to real, integer and Boolean constraints, Journal of Logic Programming 32 (1) (1997) 1–24.
[4] J. Canny, Some algebraic and geometric computations in PSPACE, in: Proc. 20th ACM Symposium on the Theory of Computing, 1988, pp. 460–467.
[5] O. Caprani, K. Madsen, Mean value forms in interval analysis, Computing 25 (1980) 147–154.
[6] J.G. Cleary, Logical arithmetic, Future Generation Computing Systems 2 (2) (1987) 125–149.
[7] O.P.D. Cutteridge, Powerful 2-part program for solution of nonlinear simultaneous equations, Electronics Letters 10 (1971).
[8] J.E. Dennis, R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice Hall, Englewood Cliffs, NJ, 1983.
[9] J.J. Ebers, J.L. Moll, Large-scale behaviour of junction transistors, IEE Proc. 42 (1954) 1761–1772.
[10] R.S. Garfinkel, G.L. Nemhauser, Integer Programming, John Wiley, New York, 1972.
[11] V. Gehrke, W. Marquardt, Direct computation of all singular points of chemical engineering models using interval methods, in: International Conference on Interval Methods and Computer Aided Proofs in Science and Engineering, Würzburg, Germany, 1996.
[12] R. Hammer, M. Hocks, M. Kulisch, D. Ratz, Numerical Toolbox for Verified Computing I—Basic Numerical Problems, Theory, Algorithms, and PASCAL-XSC Programs, Springer, Heidelberg, 1993.
[13] E. Hansen, Global Optimization Using Interval Analysis, Marcel Dekker, New York, 1992.
[14] E.R. Hansen, R.I. Greenberg, An interval Newton method, Appl. Math. Comput. 12 (1983) 89–98.
[15] E.R. Hansen, S. Sengupta, Bounding solutions of systems of equations using interval analysis, BIT 21 (1981) 203–211.
[16] E.R. Hansen, R.R. Smith, Interval arithmetic in matrix computation: part II, SIAM Journal on Numerical Analysis 4 (1967) 1–9.
[17] W. Hock, K. Schittkowski, Test examples for nonlinear programming codes, Lecture Notes in Economics and Mathematical Systems, Springer, 1981.
[18] H. Hong, V. Stahl, Safe starting regions by fixed points and tightening, Computing 53 (3–4) (1994) 323–335.
[19] J. Moré, B. Garbow, K. Hillstrom, Testing unconstrained optimization software, ACM Transactions on Mathematical Software 7 (1) (1981) 17–41.
[20] R.B. Kearfott, Preconditioners for the interval Gauss–Seidel method, SIAM Journal of Numerical Analysis 27 (1990) 804–822.
[21] R.B. Kearfott, A review of preconditioners for the interval Gauss–Seidel method, Interval Computations 11 (1991) 59–85.
[22] R.B. Kearfott, A review of techniques in the verified solution of constrained global optimization problems, to appear.
[23] R. Krawczyk, Newton-algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, Computing 4 (1969) 187–201.
[24] O. Lhomme, Consistency techniques for numerical constraint satisfaction problems, in: Proc. 1993 International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France, August 1993.
[25] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1) (1977) 99–118.

[26] A.K. Mackworth, E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, Artificial Intelligence 25 (1985) 65–74.
[27] U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, Information Science 7 (2) (1974) 95–132.
[28] R.E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1966.
[29] R.E. Moore, Methods and Applications of Interval Analysis, SIAM, Philadelphia, PA, 1979.
[30] R.E. Moore, S.T. Jones, Safe starting regions for iterative methods, SIAM Journal on Numerical Analysis 14 (1977) 1051–1065.
[31] A.P. Morgan, Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[32] A. Neumaier, Interval Methods for Systems of Equations, PHI Series in Computer Science, Cambridge University Press, Cambridge, 1990.
[33] W. Older, A. Vellino, Constraint arithmetic on real intervals, in: Constraint Logic Programming: Selected Research, MIT Press, Cambridge, MA, 1993.
[34] H. Ratschek, J. Rokne, New Computer Methods for Global Optimization, Ellis Horwood Limited, Chichester, 1988.
[35] H. Ratschek, J. Rokne, Experiments using interval analysis for solving a circuit design problem, Journal of Global Optimization 3 (1993) 501–518.
[36] J. Renegar, A faster PSPACE algorithm for the existential theory of the reals, in: Proc. 29th IEEE Symp. Foundations of Computer Science, 1988, pp. 291–295.
[37] S.M. Rump, Verification methods for dense and sparse systems of equations, in: J. Herzberger (Ed.), Topics in Validated Computations, Elsevier, Amsterdam, 1988, pp. 217–231.
[38] P. Van Hentenryck, D. McAllister, D. Kapur, Solving polynomial systems using a branch and prune approach, SIAM Journal on Numerical Analysis 34 (2) (1997) 797–827.
[39] P. Van Hentenryck, L. Michel, Y. Deville, Numerica: a Modeling Language for Global Optimization, MIT Press, Cambridge, MA, 1997.
[40] P. Van Hentenryck, V. Saraswat, Strategic directions in constraint programming, ACM Computing Surveys 28 (4) (1996).