Helios: A Modeling Language for Nonlinear Constraint Solving and Global Optimization using Interval Analysis

Pascal Van Hentenryck and Laurent Michel

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-95-33
November 1995

`Helios`: A Modeling Language for Nonlinear Constraint Solving and Global Optimization using Interval Analysis

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

Abstract

This paper introduces `Helios`, a mathematical modeling language to solve nonlinear constraint systems and global optimization problems using interval analysis. `Helios` makes it possible to state nonlinear applications almost as in scientific publications by offering computer-readable equivalents of many mathematical notations. As a consequence, `Helios` significantly simplifies the solving of these applications by letting users focus on modeling aspects instead of on low-level details. `Helios` statements are used as inputs to state-of-the-art interval analysis algorithms, which combine techniques from numerical analysis and artificial intelligence. `Helios` has been evaluated on a variety of benchmarks. On traditional constraint-solving benchmarks, it outperforms the interval methods we are aware of and is competitive with state-of-the-art continuation methods. On global optimization problems, `Helios` outperforms the interval algorithms we are aware of.

1 Introduction

Many applications in science and engineering (e.g., chemistry, robotics, economics, mechanics) require finding all isolated solutions to a system of nonlinear constraints over real numbers or finding the global optimum of an objective function subject to some nonlinear constraints. These problems are difficult due to their inherent computational complexity (i.e., it is NP-hard) and due to the numerical issues involved to ensure termination and guarantee correctness (i.e., finding all solutions). Several interesting methods have been proposed in the past for the former task, including two fundamentally different methods: interval methods (e.g., [4, 5, 7, 8, 11, 14, 15, 18, 23, 30, 35]) and continuation methods (e.g., [29, 43]). Continuation methods have been shown to be effective for problems for which the total degree is not too high, since the number of paths explored depends on the estimation of the number of solutions. Interval methods are generally robust and have been shown to be competitive with continuation methods on a variety of benchmarks [40]. Interval methods can also be applied to global optimization as advocated in numerous papers (e.g., [6, 5, 16, 26, 33]).

Interval methods have been investigated extensively in the last decades, leading to numerous theoretical results and to the implementation of many algorithms. However, few software tools are available to help engineers and scientists applying this body of knowledge. Several interval libraries (e.g., [17]) offer interval extensions of many real functions but there is a considerable gap between these libraries and practical algorithms. Some interval packages for solving systems of polynomial equations are also available (e.g., [13]). In general, these packages only require the specification of a coefficient matrix for the constraints and hence they reduce the distance between interval libraries and practical applications substantially. However, these matrices are low-level, computer-centered, specifications of nonlinear applications; they are still far from the mathematical descriptions typically found in scientific publications. Constraint programming languages based on interval analysis (e.g., `BNR-Prolog` [31], `CLP(BNR)` [3], and `Newton` [41]) are a recent addition to the set of tools supporting interval analysis. These languages also bridge some of the gap between interval libraries and nonlinear applications. In addition, some of them (e.g., `Newton` [41]) are based on state-of-the-art interval algorithms, which make them appealing for computer scientists familiar with this class of languages. Nevertheless, constraint programs are still far from traditional statements of these applications. As a consequence, there is a need for software supporting interval analysis at a higher, human-centered, level.

This paper reports a step into this direction. It describes `Helios`, a modeling language for stating and solving nonlinear applications using interval analysis. `Helios` makes it possible to state a wide range of nonlinear problems in a way which is close to traditional descriptions of these applications in scientific publications. `Helios` is an algebraic language using computer-readable equivalents of expressions such as

$$\sum_{i=1}^{n} a_i$$

or

$$s_i^2 + c_i^2 = 1 \quad (1 \leq i \leq m)$$

which are frequently used for stating nonlinear problems. Some of the important features supported by `Helios` are sets, arrays, aggregation operators such as summation and product, and generic constants, functions, and constraints. `Helios` significantly simplifies the solving of these applications by letting users focus on high-level modeling issues instead of on low-level details. Note also that `Helios` is similar in philosophy to modeling languages such as `AMPL`, although their design and implementation differ because of the targeted application areas.

`Helios` statements are used as input to state-of-the-art interval algorithms. In particular, `Helios` is based on a branch and prune algorithm described in [40] which combines techniques from numerical analysis and artificial intelligence. This branch and prune algorithm has been extended into a branch and bound algorithm for solving global optimization problems. `Helios` has been evaluated on a variety of benchmarks from kinematics, chemistry, combustion, economics, and mechanics. It outperforms the interval methods we are aware of and compares well with state-of-the-art continuation methods on many problems.

The rest of this paper is organized as follows. Section 2 gives an informal presentation of the functionalities of `Helios` and motivates some of our design decisions. Section 3 gives a tour of `Helios`, illustrating `Helios` on a number of applications. Section 4 gives an overview of the algorithms. Section 5 contains the preliminaries on interval analysis. Section 6 presents an abstract version of the branch & prune algorithm for constraint solving. Section 7 presents the branch and bound algorithm for unconstrained optimization, while Section 8 presents the branch and bound algorithm for constrained optimization. Section 9 describes the experimental results. Section 10 discusses the origins of this work and some related work. Section 11 concludes the paper.

2 Overview of `Helios`

The purpose of an `Helios` statement is to describe a set of constraints to be solved and, possibly, an objective function to optimize. `Helios` provides a number of facilities to simplify this specification without sacrificing efficiency.[1] First, `Helios` offers a number of features that are common to all algebraic languages: arrays of variables are used as the computer-readable equivalents of subscripted variables and aggregation operators such as

```
sum(i in [1..n]) a[i]
```

are used as equivalents of the mathematical expressions

$$\sum_{i=1}^{n} a_i.$$

Second, `Helios` offers generic constraints, since many applications are expressed in terms of closely related constraints. For instance, a set of constraints of the form

$$s_i^2 + c_i^2 = 1 \quad (1 \le i \le m)$$

is frequent in kinematics applications and is expressed in `Helios` as

```
trigo(i in [1..m]):  s[i]^2 + c[i]^2 = 1.
```

Third, `Helios` also supports a rich language for specifying sets. This functionality is often needed in practice for specifying the range of aggregation operators. For instance, the Broyden-banded function benchmark is typically specified using a set

$$J_i = \{j \mid \max(1, i-5) \le j \le \min(n, i+1) \ \& \ j \ne i\}$$

which is specified in `Helios` as

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

Fourth, `Helios` also offers the ability to specify constants, i.e., symbolic names that are associated with a value. For instance, in practical applications, it is frequent to find equations of the form

$$r = 0.0002155/\sqrt{40}.$$

Instead of creating a variable `r` and an equation, it is best to view `r` as a constant whose value is given by $0.0002155/\sqrt{40}$. `Helios` guarantees that the expression associated with a constant is evaluated once. Fifth, `Helios` supports user-defined functions, since the ability to specify intermediary functions may simplify the mathematical statements and may reduce the risk of errors. For instance, the Moré-Cosnard benchmark uses a function

```
p(j in idx) = (x[j]+t[j]+1)^3.
```

[1] `Helios` has a number of other features that are not reviewed here such as pragmas for controlling the algorithms.

Functions in `Helios` are simply abbreviations for complex expressions. Contrary to constants, they may be defined in terms of variables and may be evaluated many times during the solving process. Finally, `Helios` offers various ways of stating the search space in which to search for solutions. In general, the search space is simply a box associating an interval with each variable. However, for other applications, it is of benefit to specify the search space more precisely. This functionality makes it possible to remove symmetries for instance.

`Helios` statements are transformed into a set of constraints (resp. an objective function subject to a set of constraints) which are used as input to a branch and prune algorithm (resp. a branch and bound algorithm). The algorithms, which combine techniques from numerical analysis and artificial intelligence, use intervals to address the two fundamental difficulties listed above. Numerical reliability is obtained by evaluating functions over intervals using outward rounding (as in interval methods). The complexity issue is addressed by using constraints to reduce the intervals early in the search. Pruning is achieved by enforcing a unique local consistency condition, called box-consistency, at each node of the search tree.[2] The algorithms are also global which makes sure that all solutions (or all global optima) are returned.

In summary, `Helios` has the following functionalities.

- **Ease of Use:** `Helios` is a mathematical modeling language which enables scientists and engineers to state nonlinear applications almost as in technical papers.
- **Correctness:** For a system of constraints, `Helios` returns a set of boxes, i.e., tuples of intervals of required accuracy; these boxes are guaranteed to contain all solutions to the constraints. In addition, for many applications, `Helios` proves that each box contains a unique solution. For optimization problems, `Helios` returns an interval bounding the value of the global optima. It also returns a set of boxes containing all the global optima.
- **Termination:** `Helios` always terminates.
- **Effectiveness:** On a variety of benchmarks taken from papers on continuation methods, interval analysis, and numerical analysis, `Helios` outperforms the interval methods we are aware of and compares well with state-of-the-art continuation methods on many problems. Interestingly, `Helios` solves the Broyden banded function problem [8] and Moré-Cosnard discretization of a nonlinear integral equation [27] for several hundred variables.

3 A Tour of Helios

This section gives a gentle introduction to `Helios` through a number of examples. Its purpose is to describe informally the syntax of `Helios`, as well as its functionality and performance on a number of well-known problems. We start with some constraint-solving examples, continue with unconstrained and constrained optimization problems and conclude with a discussion of some additional facilities.

[2]Box-consistency is an approximation of arc-consistency, a notion well-known in artificial intelligence [20, 22] and used to solve discrete combinatorial problems in several systems (e.g., [38, 39]).

3.1 Constraint Solving in Helios

We start with a simple univariate problem which consists of finding the roots of the function

$$x^4 - 12x^3 + 47x^2 - 60x$$

in the interval $[-10^8, 10^8]$. The Helios statement to solve this problem is as follows:

```
Variable:
   x in [-10^8..10^8];
Body:
   solve system
      EQ: x^4 - 12 * x^3 + 47 * x^2 - 60 * x = 0;
```

The variable section declares a single variable which ranges over $[-10^8, 10^8]$. The body section specifies the system of constraints to be solved by Helios, i.e., the single constraint EQ. On this problem, Helios returns the four intervals

```
x in [0.00000000,0.00000000]
x in [2.99999999,3.00000001]
x in [4.00000000,4.00000000]
x in [4.99999999,5.00000001]
```

with the default precision of the system. Helios guarantees that all solutions to the equation are inside these intervals, i.e., a solution cannot be located outside these intervals. The compilation time for this problem is about 0.2 second (on a SUN SPARC 10), while the execution is less than 0.1 second. Note also that for the function

$$x^4 - 12x^3 + 47x^2 - 60x + 24$$

Helios returns the intervals

```
x in [0.88830577,0.88830578]
x in [0.99999999,1.00000000]
```

while Helios does not return any solution for the function

$$x^4 - 12x^3 + 47x^2 - 60x + 24.1$$

indicating the absence of solutions. Note that it is also possible to ask Helios to prove the existence and unicity of solutions. The statement simply becomes

```
Variable:
   x in [-10^8..10^8];
Body:
   unique solve system
      EQ: x^4 - 12 * x^3 + 47 * x^2 - 60 * x = 0;
```

where the keyword `unique` has been added. With this new statement, `Helios` guarantees that any interval returned as a solution contains a unique solution. Note that the proof of existence and unicity is obtained numerically and that `Helios` produces the same results for the above root-finding problems. For all the examples in this section, `Helios` is able to prove the existence and unicity of the solutions. However, in general, `Helios` may find some intervals that are unique, i.e., intervals which contain a unique solution, and some other intervals for which no conclusion can be drawn, i.e., intervals of small size for which `Helios` cannot prove the existence or absence of a solution because of the precision of the floating-point system.

We now consider a simple multivariate problem: the intersection of a circle and a parabola. The problem can be stated as follows:

```
Variable:
   x :  array[1..2];
Body:
   unique solve system
      Circle:    x[1]^2 + x[2]^2 = 1;
      Parabola:  x[1]^2 - x[2] = 0;
```

The variable section declares an array of two variables and the body section specifies the two constraints. `Helios` returns the two boxes (i.e., two tuples of intervals)

```
x[1] in [-0.78615138,-0.78615137]
x[2] in [+0.61803398,+0.61803399]

x[1] in [+0.78615138,+0.78615137]
x[2] in [+0.61803398,+0.61803399]
```

in about 0.1 second.

Some more interesting multivariate problems come from the area of 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. The `Helios` statement to solve a problem given in [11] is depicted in Figure 1. There are several novel features in the statement. First, the set section declares a set `idx` that is used subsequently to define arrays and to specify constraints. Second, the example illustrates how constraints can be stated generically. The `Helios` statement generates trigonometric constraints of the form

$$\mathtt{s[i]}^2 + \mathtt{c[i]}^2 = 1$$

for $1 \leq \mathtt{i} \leq 6$. The compilation of this program takes about 0.7 second and the running time to generate all solutions with the above statement is about 30 seconds. If only one solution is needed, the keyword `once` can be inserted after the `unique solve system` and only a single solution is produced (in about 2 seconds).

Chemistry is the source of many interesting nonlinear problems and our next application of `Helios` is a chemical equilibrium problem for the propulsion of propane into the air. The problem is taken from [21] and the `Helios` statement is depicted in Figure 2. It illustrates the use of

```
Set:
   idx = [1..6];

Variable:
   s :  array[idx] in [-10^8..10^8];
   c :  array[idx] in [-10^8..10^8];

Body:  unique solve system
   trigo(i in idx) :  s[i]^2 + c[i]^2 = 1;
   C1 :  s[2]*c[5]*s[6] - s[3]*c[5]*s[6] - s[4]*c[5]*s[6] +
         c[2]*c[6] + c[3]*c[6] + c[4]*c[6] = 0.4077;
   C2 :  c[1]*c[2]*s[5] + c[1]*c[3]*s[5] + c[1]*c[4]*s[5] + s[1]*c[5] = 1.9115;
   C3 :  s[2]*s[5] + s[3]*s[5] + s[4]*s[5] = 1.9791;
   C4 :  c[1]*c[2] + c[1]*c[3] + c[1]*c[4] + c[1]*c[2] + c[1]*c[3] + c[1]*c[2] = 4.0616;
   C5 :  s[1]*c[2] + s[1]*c[3] + s[1]*c[4] + s[1]*c[2] + s[1]*c[3] + s[1]*c[2] = 1.7172;
   C6 :  s[2] + s[3] + s[4] + s[2] + s[3] + s[2] = 3.9701;
```

Figure 1: A Robot Kinematics Application in Helios.

```
Variable:
   y :  array[1..5] in [0..10^8];

Constant:
   r = 10;
   r5 = 0.193;
   r6 = 0.002597/sqrt(40);
   r7 = 0.003448/sqrt(40);
   r8 = 0.00001799/40;
   r9 = 0.0002155/sqrt(40);
   r10 = 0.00003846/40;

Body:  unique solve system
   C1:  0 = 3*y[5] - y[1]*(y[2] + 1);
   C2:  0 = y[2]*(2*y[1] + y[3]^2 + r8 + 2*r10*y[2] + r7*y[3] + r9*y[4]) + y[1] - r*y[5];
   C3:  0 = y[3]*(2*y[2]*y[3] + 2*r5*y[3] + r6 + r7*y[2]) - 8*y[5];
   C4:  0 = y[4]*(r9*y[2] + 2*y[4]) - 4*r*y[5];
   C5:  0 = y[2]*(y[1] + r10*y[2] + y[3]^2 + r8 + r7*y[3] + r9*y[4]) +
            y[1] + r5*y[3]^2 + y[4]^2 - 1 + r6*y[3];
```

Figure 2: A Chemical Equilibrium Application in Helios.

```
Input:
   int n :  "Number of variables:  ";

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

Variable:
   x :  array[idx] in [-10^8..10^8];

Body:  unique solve system
   f(i in idx):
      0 = x[i] * (2 + 5 * x[i]^2) + 1 - Sum(j in sidx(i)) x[j] * (1 +x[j]);
```

Figure 3: The Broyden Banded Function in `Helios`

n	Compilation Time (ms)	Running Time (ms)	Growth Factor
5	290	350	
10	290	1860	5.31
20	290	6150	3.30
40	290	15360	2.49
80	290	38730	2.52
160	290	105610	2.72

Figure 4: Performance Results of `Helios` on the Broyden Banded function.

constants in `Helios`. A constant gives a symbolic name to an expression; the name can then be used in specifying constraints. Constants are assumed to denote real numbers unless specified otherwise by an integer declaration. `Helios` guarantees that a constant is evaluated only once. The compilation time for this example is about 0.6 second and the running time to obtain the unique solution

y[1] in [0.00311409, 0.00311411]
y[2] in [34.59792453, 34.59792454]
y[3] in [0.06504177, 0.06504178]
y[4] in [0.85937805, 0.85937806]
y[5] in [0.03695185, 0.03695186]

is about 5 seconds.

Our next example is the well-known Broyden Banded function problem (e.g., [8]) which amounts to finding the zeros of the system of n equations

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

where $J_i = \{j \mid \max(1, i-5) \leq j \leq \min(n, i+1) \ \& \ j \neq i\}$. The `Helios` statement depicted in Figure 3 follows closely the mathematical statement. There are several novelties that are worth discussing. The first novelty in this statement is the input section, which allows the user to specify constants at runtime. When `Helios` executes the above statement, it queries the user for the value of `n` to obtain the number of variables. This facility is particularly useful for problems where the dimension is not fixed. The second novelty is the use of a parametric set which defines the set J_i in the above mathematical statement. The set `sidx` is parametrized by an argument `i` and it defines a set for each `i` in `idx`. Note that the set is defined by first restricting `j` to lie in the interval `[max(1,i-5)..min(n,i+1)]` and then by using the filter `j <> i` to restrict the values in the interval. The last novelty is the use of the `Sum` operator in the constraint statement. The operator takes a subscript that must range inside a set which is either defined in the set section (as it is the case here) or which is given inline. The example illustrates nicely that `Helios` statements can be very close to the statements usually published in scientific papers. Another interesting fact is that `Helios` is essentially linear in the number of variables on this benchmark. The performance results are given in Figure 4.

Our last example of equation solving is another traditional benchmark from numerical analysis: the discretization of a nonlinear integral equation [27]. The objective is to find the zeros of the functions $f_k(x_1, \ldots, x_m)(1 \leq k \leq m)$ defined as follows:

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

with $t_j = jh$ and $h = 1/(m+1)$. The `Helios` statement is depicted in Figure 5 and illustrates some new constructs as well. First, the statement shows the use of an array of constants (i.e., `t`) defined by a generic expression (i.e., `j * h`). These are called generic constants. Second, the statement illustrates the use of a function `p` which is an abbreviation for a complex expression. Note that functions can be defined in terms of variables contrary to constants. As a result, the `Helios` statement is once again very close to the traditional published description of the problem. The performance results are given in Figure 6 and indicates that `Helios` is between linear and quadratic in the size of the constraint system (i.e., the square of the number of variables) and between quadratic and cubic in the number of variables.

3.2 Unconstrained Optimization in `Helios`

We now turn to optimization problems and consider first unconstrained optimization. Our first example is the Rosenbrock function [19] which consists of minimizing the function

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

The `Helios` statement is depicted in Figure 7. It contains the keyword `minimize` followed by the objective function to be minimized (instead of the keyword `solve system` followed by the

```
Input:
   int m :  "Give the number of variables:  ";

Set:
   idx = [1..m];

Variable:
   x :  array[idx] in [-4..5];

Constant:
   h = 1/(m+1);
   t[j in idx] = j * h;

Function:
   p(j in idx) = (x[j]+t[j]+1)^3;

Body:  unique solve system
   f(k in idx):
      0 = 2 * (m+1) * x[k] +
          (1 - t[k]) * Sum(j in [1..k]) t[j]*p(j) +
          t[k]       * Sum(j in [k+1..m]) (1-t[j])*p(j);
```

Figure 5: The Moré-Cosnard Benchmark in `Helios`.

n	n^2	Compilation Time (ms)	Running Time (ms)	Growth Factor
5	25	480	1190	
7	49	480	3010	2.52
10	100	480	8350	2.77
14	196	480	20730	2.48
20	400	480	59490	2.86
28	784	480	176960	2.97
40	1600	480	535700	3.02

Figure 6: Performance Results of `Helios` on Moré-Cosnard Problem.

```
Set:
   idx = [1..2];
Variable:
   x :  array[idx] in [-10^8..10^8];
Body:  minimize
   100 * (x[2] - x[1]^2)^2 + (x[1] - 1)^2;
```

Figure 7: The Rosenbrock Function in Helios.

```
Input:
   int n :  "Number of variables";

Set:
   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);
```

Figure 8: Unconstrained Optimization in Helios: Problem Levy 5.

n	Compilation Time (ms)	Running Time (s)	Growth Factor
5	230	1.04	
10	230	3.34	3.21
20	230	11.82	3.53
40	230	45.89	3.88

Figure 9: Performance Results of Helios on Problem Levy 5.

```
Set:
   idx = [1..2];

Variable:
   x :  array[idx] in [-10..10];

Body:
   minimize
      Prod(k in [1..2]) (Sum(i in [1..5]) i * cos((i+1)*x[k] + i));
```

Figure 10: Unconstrained Optimization in `Helios`: Problem `Levy 3`.

constraint system to be solved). On unconstrained optimization problems, `Helios` returns an interval bounding the value of the global optimum as well as boxes enclosing each of the optima. For instance, in the `Rosenbrock` problem, `Helios` returns

optimum	in	[0.00000000, 0.00000001]
x[1]	in	[0.99999999, 1.00000001]
x[2]	in	[0.99999999, 1.00000001]

The compilation and running times of `Helios` are 0.15 and 0.4 second respectively. It is important to stress that `Helios` bounds the global optimum value, not a local minimum value. In addition, `Helios` returns all the global optima. A nice example from [19] illustrating this fact is the minimization of the function

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

For $n = 10$, the function has 10^{10} local minima but a single global optimum. Figure 8 depicts the `Helios` statement which involves several of the features of the languages: input constant, minimization, function, and summation. In addition, it uses a trigonometric function (i.e., `sin`) and the predefined constant `pi`. `Helios` seems to be essentially quadratic in the number of variables on this problem as shown in Figure 9. Another example is the minimization of the function

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

which has 760 local minima and 18 global minima in [-10,10]. The `Helios` statement is shown in Figure 10 and it illustrates the product operator. Note that `Helios` returns boxes for all 18 global minima in about 20 seconds.

3.3 Constrained Optimization in `Helios`

Constrained optimization problems can also be stated in `Helios`. Figure 11 depicts the `Helios` statement of a constrained optimization problem taken from [10]. The main difference with uncon-

```
Variable:
   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];
Constant:
   b1 = 4.97;
   b2 = -1.88;
   b3 = -29.08;
   b4 = -78.02;
Body:
   minimize
      4.3 * x1 + 31.8 * x2 + 63.3 * x3 + 15.8 * x4 + 68.5 * x5 + 4.7 * x6
   subject to
      C1 :  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;
      C2 :  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;
      C3 :  -273 * x2 - 70 * x4 - 819 * x5 + 26000 * x4 * x5 >= b3;
      C4 :  159.9 * x1 - 311 * x2 + 587 * x4 + 391 * x5 + 2198 * x6
            - 14000 * x1 * x6 >= b4;
```

Figure 11: Constrained Optimization in Helios.

strained optimization is the keyword `subject to` which is followed by the description of the constraints to be satisfied. Once again, Helios is guaranteed to return an interval enclosing the value of the global optimum as well as boxes enclosing all the global optima. The keyword `optimistic` can also be added, in which case Helios is optimistic and simply assumes that each canonical box that is not pruned away by the constraints contains a solution.[3]

3.4 Additional Functionalities

Helios has a number of additional functionalities besides those described previously. For instance, Helios has a number of pragmas to influence the constraint solver (e.g., to specify the width of the intervals).

Helios also has some facilities to specify the range of variables. In the examples presented so far, each variable was given a range specified by an interval. Helios also makes it possible to specify the range of the variables using constraints. Range constraints specify the region in which

[3]Note that, in unconstrained optimization, no such issue arises since there are no constraints.

```
Set:
   idx = [1..2];

Variable:
   x :  array[idx] in [-10..10];

Body:
   minimize
      Prod(k in [1..2]) (Sum(i in [1..5]) i * cos((i+1)*x[k] + i))
   with range constraints
      x[1] >= x[2];
```

Figure 12: Unconstrained Optimization in Helios: Problem Levy 3 with Range Constraints.

```
Set:
   idx = [1..6];

Variable:
   s :  array[idx] in [-10^8..10^8];
   c :  array[idx] in [-10^8..10^8];

Body:  solve system once
   trigo(i in idx) :  s[i]^2 + c[i]^2 = 1;
   C1 :  s[2]*c[5]*s[6] - s[3]*c[5]*s[6] - s[4]*c[5]*s[6] +
         c[2]*c[6] + c[3]*c[6] + c[4]*c[6] = 0.4077;
   C2 :  c[1]*c[2]*s[5] + c[1]*c[3]*s[5] + c[1]*c[4]*s[5] + s[1]*c[5] = 1.9115;
   C3 :  s[2]*s[5] + s[3]*s[5] + s[4]*s[5] = 1.9791;
   C4 :  c[1]*c[2] + c[1]*c[3] + c[1]*c[4] + c[1]*c[2] + c[1]*c[3] + c[1]*c[2] = 4.0616;
   C5 :  s[1]*c[2] + s[1]*c[3] + s[1]*c[4] + s[1]*c[2] + s[1]*c[3] + s[1]*c[2] = 1.7172;
   C6 :  s[2] + s[3] + s[4] + s[2] + s[3] + s[2] = 3.9701;
Display:
   variables:  c,s;
   constraints :  C1,C2,C3,C4,C5,C6;
```

Figure 13: A Robot Kinematics Application in Helios with Display Section.

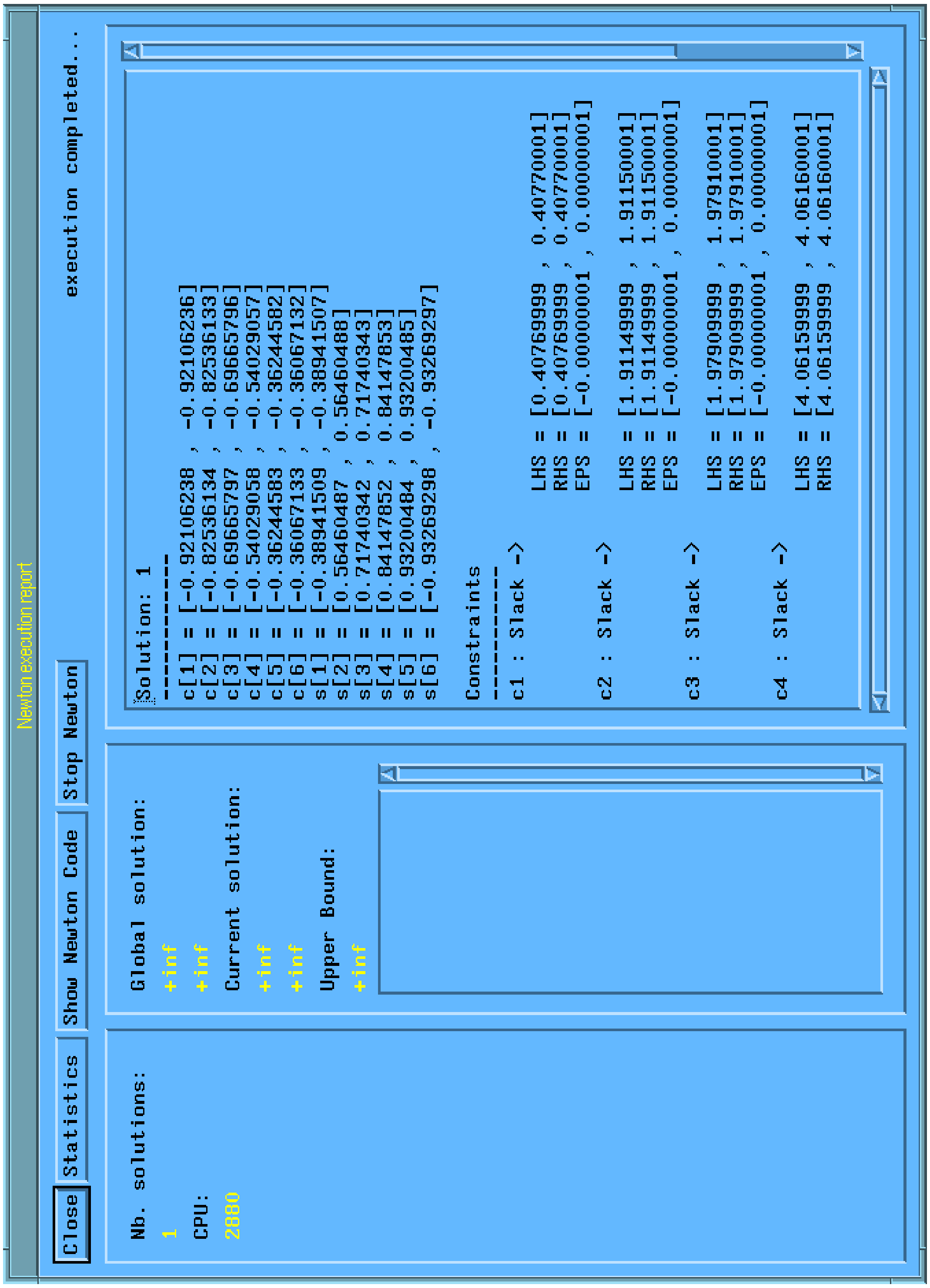

Figure 14: The Display of the Solution to the Kinematics Problem.

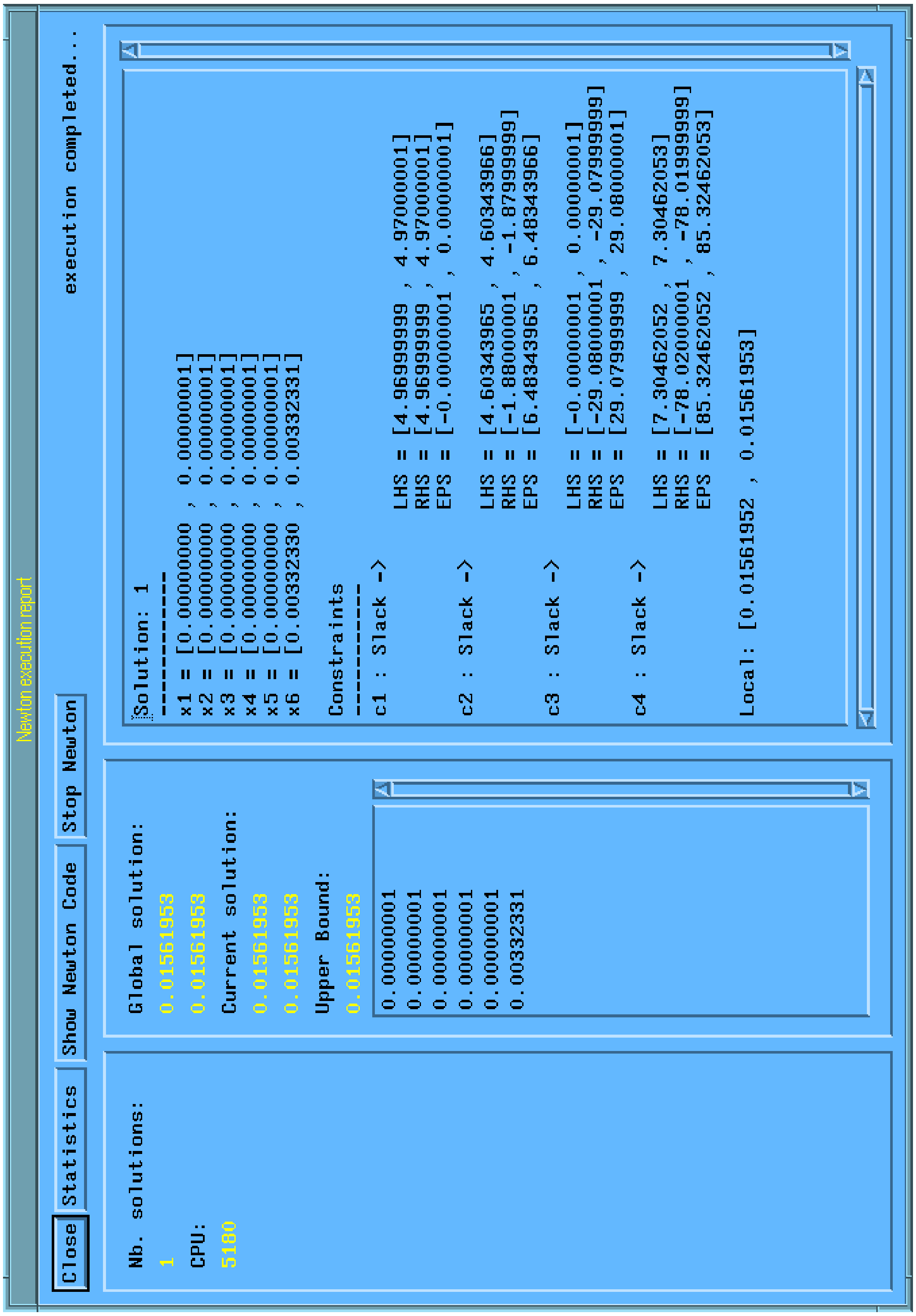

Figure 15: The Display of the Solution to Problem h95.

to search for solutions: they are not used for proving the existence of solutions.[4] A new `Helios` statement for `Levy 3`, including range constraints, is depicted in Figure 12. The range constraint exploits the fact that the variables play a symmetric role and constrains `x[1]` to be no smaller than `x[2]`. The constraint reduces the execution time by a factor 2.

Finally, there is another functionality that is worth mentioning: the display section. The display section makes it possible to specify the information to be produced. Figure 13 describes the `Helios` statement for the kinematics example with a display section. The display section requests to display the variables as well as the constraints `C1,...,C6`. The output is depicted in Figure 14. An interval is associated with each variable. In addition, three intervals are associated with each constraint, bounding the left-hand side, the right-hand side, and their difference. Note that displaying the constraints provides users with information on numerical accuracy and makes it possible to detect badly conditioned or badly formulated problems. It suffices to examine the size of the intervals associated with the constraints. Figure 15 describes the output of the constrained optimization problem `h95`. In optimization problems, the information displayed also allows users to find which inequalities are tight at optimality. In `h95`, for instance, the first constraint is tight, while the remaining ones are not.

4 An Overview of the Algorithms

The purpose of this section is to give readers an informal understanding of the algorithms before the subsequent more technical sections. As mentioned, the kernel of `Helios` is a global search algorithm (fully described in [40]) which solves a constraint solving problem by dividing it into subproblems which are solved recursively. More precisely, `helios` uses a branch and prune algorithm which is best viewed as an iteration of two steps

1. pruning the search space;

2. making a nondeterministic choice to generate two subproblems

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

The pruning step ensures that some local consistency conditions are satisfied. It consists of reducing the intervals associated with the variables so that every constraint appears to be locally consistent. The local consistency condition of `Helios` is called box-consistency, an approximation of arc-consistency, a notion well-known in artificial intelligence [20, 22] and used in many systems (e.g., [39, 42, 37]) to solve discrete combinatorial search problems. Informally speaking, a constraint is arc-consistent if for each value in the range of a variable there exist values in the ranges of the other variables such that the constraint is satisfied. `Helios` approximates arc-consistency which cannot be computed on real numbers in general.

The pruning step either fails, showing the absence of solution in the intervals, or enforces the local consistency condition. Sometimes, local consistency also implies global consistency as in the case of the Broyden banded function presented previously. The pruning step of `Helios` solves this problem in essentially linear time for initial intervals of the form $[-10^8, 10^8]$ and always proves the existence of a solution in the final box. However, in general, local consistency does not imply global

[4] Users are responsible to ensure that solutions (resp. global solutions) exist in the search region.

consistency either because there are multiple solutions or simply because the local consistency condition is too weak. Consider the intersection of a circle and of a parabola:

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

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

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

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

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

These zeros are -1 and 1 and hence the new interval for x_1 becomes [-1,1] (modulo the numerical precision).[5] The same reasoning applies to x_2. The second constraint can be used to reduce further the interval of x_2 by searching for the leftmost and rightmost zeros of

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

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

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

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

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

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

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

Backtracking on the choice for x_1 produces the intervals

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

and to the second solution

[5] The precision of the pruning step depends on the numerical precision and on some parameters of the system, which explains the digits 12430057 in the above results.

$$
\begin{aligned}
x_1 &\in [+0.7861513777574231642, +0.7861513777574233864] \\
x_2 &\in [+0.6180339887498946804, +0.6180339887498950136].
\end{aligned}
$$

Note that, in this case, Helios makes the smallest number of choices to isolate the solutions. Let us illustrate the pruning of Helios on a larger example: the kinematics example presented previously. The pruning step returns the intervals

$$
\begin{aligned}
s_1 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
c_1 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
s_2 &\in [+0.3233666666666665800, +1.0000000000000000000] \\
c_2 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
s_3 &\in [-0.0149500000000000189, +1.0000000000000000000] \\
c_3 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
s_4 &\in [-0.0209000000000001407, +1.0000000000000000000] \\
c_4 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
s_5 &\in [+0.6596999999999998420, +1.0000000000000000000] \\
c_5 &\in [-0.7515290480087772896, +0.7515290480087772896] \\
s_6 &\in [-1.0000000000000000000, +1.0000000000000000000] \\
c_6 &\in [-1.0000000000000000000, +1.0000000000000000000]
\end{aligned}
$$

showing already some interesting pruning. After exactly 12 branchings and in less than a second, Helios produces the first box with a proof of existence of a solution in the box. Consider now an optimization problem such as the minimization of the function

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

Helios applies the branch and prune algorithm to the necessary conditions

$$
S = \{\frac{\partial f}{\partial x_1} = 0, \frac{\partial f}{\partial x_2} = 0, f \leq u\}
$$

where u is an upper bound on the value of the global optima. Initially, u is ∞ but it is updated dynamically during the algorithm, each time a new upper bound is found. New upper bounds are obtained by probing some values in the interval considered at some given stage. For instance, the first upper bound obtained by Helios on the above problem is 19.87584009 by probing at point $(0, 0)$. Constrained optimization problems are solved using the same ideas, except that the optimality conditions and the probing techniques are more complicated.

5 Interval Analysis

In this section, we review some basic concepts needed for this paper, including interval arithmetic and the representation of constraints. More information on interval arithmetic can be found in many places (e.g., [1, 8, 7, 23, 24]). Our definitions are slightly non-standard.

5.1 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.[6]

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

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

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

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

Definition 3 [Interval Extension] $F : \mathcal{I}^n \rightarrow \mathcal{I}$ is an interval extension of $f : \Re^n \rightarrow \Re$ 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 $C : \mathcal{I}^n \rightarrow \mathit{Bool}$ is an interval extension of a relation $c : \Re^n \rightarrow \mathit{Bool}$ 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.

[6]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 `Helios` operators and relations (for instance, the interval extension of $+$ is defined by $\oplus$). In addition, we overload the real symbols and use them for their interval extensions. Finally, we denote relations by the letter c possibly subscripted, interval relations by the letter C possibly subscripted. Note that constraints and relations are used as synonyms in this paper.

5.2 Constraint Representations

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

6 Constraint Solving

This section gives a high-level overview of the branch & prune algorithm of `Helios`. It is based on [40], where all details can be found. Section 6.1 defines box-consistency, the key concept behind our algorithm. Section 6.2 shows how box-consistency can be instantiated to produce various pruning operators achieving various tradeoffs between accuracy and efficiency. Section 6.3 defines a conditioning operator used in `Helios` to improve the effectiveness of box-consistency. Section 6.4 specifies the pruning in `Helios`. Section 6.5 describes the algorithm. Recall that we assume that all constraints are defined over variables $x_1, \ldots, x_n$.

6.1 Box Consistency

Box-consistency [2] is an approximation of arc-consistency, a notion well-known in artificial intelligence [20] 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 $D_i \subseteq \{r_i \mid \exists r_1 \in D_1, \ldots, \exists r_{i-1} \in D_{i-1}, \exists r_{i+1} \in D_{i+1}, \ldots, \exists r_n \in D_n : c(r_1, \ldots, r_n)\}$. A constraint c is arc-consistent wrt $\langle D_1, \ldots, D_n\rangle$ if each of its projections is arc-consistent wrt $\langle D_1, \ldots, D_n\rangle$. A system of constraints $\mathcal{S}$ is arc-consistent wrt $\langle D_1, \ldots, D_n\rangle$ if each constraint in $\mathcal{S}$ is arc-consistent wrt $\langle D_1, \ldots, D_n\rangle$.

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

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

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

This condition, used in systems like [31, 3], is easily enforced on simple constraints such as

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

but it is also computationally very expensive for complex constraints with multiple occurrences of the same variables. Moreover, decomposing complex constraints into simple constraints entails a substantial loss in pruning, making this approach unpractical on many applications. See [2] for experimental results on this approach and their comparison with the approach presented in this paper.

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

Definition 6 [Interval Projection Constraint] An interval projection constraint $\langle C, i\rangle$ is the association of an interval constraint C and of an index i $(1 \le i \le n)$. Interval projection constraints are denoted by the letter P, possibly subscripted.

Definition 7 [Box-Consistency] An interval projection constraint $\langle C, i\rangle$ is box-consistent wrt $\vec{I} = \langle I_1, \ldots, I_n\rangle$ iff

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

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

Intuitively speaking, the above condition states that the ith interval cannot be pruned further using the unary interval constraint obtained by replacing all variables but X_i by their intervals, since the boundaries satisfy the unary constraint. Note also that the above condition is equivalent to

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

which shows clearly that box-consistency is an approximation of arc-consistency.[7] 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.

6.2 Interval Extensions for Box Consistency

Box-consistency strongly depends on the interval extensions chosen for the constraints and different interval extensions can produce very different (often incomparable) tradeoffs between pruning and computational complexity. In this section, we consider two extensions used in `Helios`: the natural interval extension and the Taylor interval extension.[8]

6.2.1 Natural Interval Extension

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

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

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. Using the natural extension allows users to minimize the problem of dependency of interval arithmetic and hence to increase precision.

6.2.2 Taylor Interval Extension

The second 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 [23]

[7] It is interesting to note that this definition is also related to the theorem of Miranda [30]. In this case, box-consistency can be seen as replacing universally quantified variables by the intervals on which they range.

[8] `Helios` used a third interval extension but it is only a way to speed up the computation.

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 8 [Taylor Interval Extension] Let c be a constraint $f = 0$, f be a function with continuous partial derivatives, $\vec{I}$ be a box $\langle I_1, \ldots, I_n \rangle$, 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 [32].

6.3 Conditioning

It is interesting to note that box-consistency on the Taylor interval extension is closely related to the Hansen-Segupta's operator [8], which is an improvement over Krawczyk's operator [18]. Hansen and Smith [9] also argued that these operators are more effective for a system $\{f_1 = 0, \ldots, f_n = 0\}$ wrt a box $\langle I_1, \ldots, I_n \rangle$ when the interval Jacobian

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

is diagonally dominant, i.e.,

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

where

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

They also suggest a conditioning which consists in multiplying the linear relaxation by a real matrix which is the inverse of the matrix obtained by taking the center of $M_{i,j}$. The resulting system is generally solved through Gauss-Seidel iterations, giving the Hansen-Segupta's operator. See also [14, 15] for an extensive coverage of conditioners. `Helios` exploits this idea to improve the effectiveness of box-consistency on the Taylor interval extension. The conditioning of `Helios` is abstracted by the following definition.

Definition 9 [Conditioning] Let $S = \{f_1 = 0, \ldots, f_n = 0\}$. A conditioning of S is a system $S' = \{f'_1 = 0, \ldots, f'_n = 0\}$ where

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

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

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

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

where

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

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

6.4 Pruning in `Helios`

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

Definition 10 [Box-Narrowing] Let $\langle C, i\rangle$ be an interval projection constraint and $\langle I_1, \ldots, I_n\rangle$ be a box. The narrowing operator `BOX-NARROW` is defined as

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

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

An implementation of `BOX-NARROW` using the interval Newton method is described in [40]. We are now in a position to define the pruning algorithm which consists essentially of applying the narrowing operators of each projection until no further reduction occurs. The pruning algorithm is depicted in Figure 16. It first applies box-consistency on the natural extension until no further reduction occurs and then applies box-consistency on the Taylor extension. The two steps are iterated until a fixpoint is reached. Termination of the algorithm is guaranteed since the set $\mathcal{F}$ is finite and thus the intervals can only be reduced finitely often.

6.5 The Branch and Prune Algorithm

Figure 17 is a very high level description of the branch and prune algorithm, highlighting the control flow. The algorithm applies operation `PRUNE` on the initial box. If the resulting box is empty (i.e., one of its components is empty), then there is no solution. If the resulting box is small enough (specified by the desired accuracy in solutions), then it is included in the result. The function `BRANCH` splits the box into two subboxes along one dimension (variable). Variables for splitting are chosen by `BRANCH` using a round-robin heuristic: if $\{x_1, \ldots, x_n\}$ is the set of variables, then the algorithm splits the variables in the order $x_1, x_2, \ldots, x_n$ and reiterates the process until the box is small enough.

```
procedure PRUNE(in S: Set of Constraint; inout I⃗ : I^n)
begin
      repeat
            I⃗_p := I⃗;
            BOX-PRUNE({⟨ĉ, i⟩ | c ∈ S & 1 ≤ i ≤ n}, I⃗);
            BOX-PRUNE({⟨c^{t(I⃗)}, i⟩ | c ∈ cond(S, I⃗) & 1 ≤ i ≤ n}, I⃗);
      until I⃗ = I⃗_p;
end

procedure BOX-PRUNE(in P: Set of Interval Projection Constraint; inout I⃗ : I^n)
begin
      repeat
            I⃗_p := I⃗;
            I⃗ := ⋂{BOX-NARROW(P, I⃗) | P ∈ P }
      until I⃗ = I⃗_p;
end
```

Figure 16: Pruning in Helios

```
function BranchAndPrune(S: Set of Constraint; I⃗_0 : I^n): Set of I^n;
begin
   I⃗ := PRUNE(S, I⃗_0);
   if ¬ IsEmpty(I⃗) then
      if IsSmallEnough(I⃗) then
         return {I⃗}
      else
         ⟨I⃗_1, I⃗_2⟩ := BRANCH(I⃗);
         return BranchAndPrune(S, I⃗_1) ∪ BranchAndPrune(S, I⃗_2)
      endif
   else
      return ∅
   endif
end
```

Figure 17: The Branch and Prune Algorithm

procedure `BranchAndBound(`**in** $f : \Re^n \rightarrow \Re$; **in** $\vec{I_0} : \mathcal{I}^n$; **out** $I_f : \mathcal{I}$; **out** $\mathcal{S}_g$: `Set of` $\mathcal{I}^n$`);`
begin
u `:=` ∞;
$\mathcal{S}$ `:= BranchPruneAndProbe(`$\{\frac{\partial f}{\partial x_1} = 0, \ldots, \frac{\partial f}{\partial x_n} = 0\}$, f, u, $\vec{I_0}$`);`
I_f `:=` $[\min\{left(\widehat{f}(\vec{I})) \mid \vec{I} \in \mathcal{S}\}, \min\{right(\widehat{f}(\vec{I})) \mid \vec{I} \in \mathcal{S}\}]$;
S_g `:=` $\{\vec{I} \in \mathcal{S} \mid \widehat{f}(\vec{I}) \cap I_f \neq \emptyset\}$
end

Figure 18: The Branch and Bound Algorithm for Unconstrained Optimization

6.6 Existence of Solution

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

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

where $m_i = center(I_i)$. If

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

then there exists a unique zero in $\langle I'_1, \ldots, I'_n \rangle$. A proof of this result can be found in [30] where credit is given to Moore and Nickel.

7 Unconstrained Optimization

`Helios` uses a branch and bound algorithm for solving unconstrained optimization problems. The branch and bound algorithm is closely related to previous interval-based algorithms (e.g., [5, 16]) for global optimization. The main difference comes from the constraint-solving algorithm in `Helios` which is more effective in many cases than the algorithms traditionally used.

The algorithm is also very close to the branch and prune algorithm. Assuming that f is a function to minimize, `Helios` applies the pruning step to the system of equations

$$S = \{\frac{\partial f}{\partial x_1} = 0, \ldots, \frac{\partial f}{\partial x_n} = 0\}$$

augmented by a constraint

$$f \leq u$$

where u is an upper bound of the value of the global minima. The upper bound is updated by a procedure that probes the value of the function in the middle of the intervals.

Figure 18 depicts the algorithm in the case of a minimization. The algorithm initializes the upper bound to infinity and calls the function `BrandPruneAndProbe` which returns a set of boxes

```
function BranchPruneAndProbe(S:Set of Constraint; f : ℜ^n → ℜ; u:float; I0 : I^n):Set of I^n;
begin
   I := PRUNEandPROBE(S,f,u,I0);
   if ¬ IsEmpty(I) then
      if IsSmallEnough(I) then
         return {I}
      else
         ⟨I1, I2⟩ := BRANCH(I);
         return BranchPruneAndProbe(S,f,u,I1) ∪ BranchPruneAndProbe(S,f,u,I2)
      endif
   else
      return ∅
   endif
end
```

Figure 19: The Branch, Prune and Probe Algorithm

```
procedure PRUNEandPROBE(in S:  Set of Constraint; in f : ℜ^n → ℜ;
                        inout u:float; inout I : I^n);
begin
    repeat
        Ip := I;
        PROBE(f,I,u);
        PRUNE(S ∪ {f ≤ u},I)
    until I = Ip
end

procedure PROBE(in f : I^n → I; in I : I^n; inout u:float);
begin
    u := min(u, right(f̂(center(I))))
end
```

Figure 20: The Prune and Probe Procedure

containing all global minima. The algorithm then computes an interval I_f bounding the value of the global minima. This interval is obtained by evaluating f on all boxes in $\mathcal{S}$ and taking the minimum of the lower bounds and the minimum of the upper bounds. The set of boxes containing all global minima is then obtained by removing boxes which cannot contain the global minima. The removed boxes come from the results obtained early in the computation when the upper bound u was not yet precise enough. Function `BrandPruneAndProbe` is described in Figure 19. It is very similar to the branch and prune algorithm. The only difference is the call to `PRUNEandPROBE` which replaces the call to `PRUNE`. Procedure `PRUNEandPROBE` is depicted in Figure 20. It iterates probing and pruning until no pruning takes place. Note that pruning takes into account the inequality $f \leq u$. Probing considers the value of the function for the center of the current interval and possibly updates the upper bound if a better value has been found.[9]

8 Constrained Optimization

Constrained optimization problems follow essentially the same line as unconstrained optimization problems. There are two significant differences however:

1. The optimality conditions such as the Fritz-John conditions (e.g., [5]) involve both the constraint system and the objective function. In addition, they introduce new variables.

2. Probing is not as easy, since the constraints must be satisfied.

When the constraint system consists only of inequalities, probing can still be performed by using the center of the intervals. It is however necessary to verify that the point satisfies the inequalities. In presence of equations, probing is essentially impossible. However, techniques for proving the existence of a solution in a box (such as those of Section 6.6) can be used (see for instance [5, 16] for more discussion of these techniques). If it can be shown that $\vec{I}$ contains a solution, then $right(\widehat{f}(\vec{I}))$ may be used as an upper bound. The probing procedure then becomes

procedure `PROBE(`**in** $f : \Re^n \to \Re$; **in** $\vec{I} : \mathcal{I}^n$; **in** $\mathcal{S}$: `Set of Constraint;` **inout** u`:float);`
begin
    if $center(\vec{I})$ `is a solution to` $\mathcal{S}$ **then**
        u `:=` $\min(u, right(\widehat{f}(center(\vec{I}))))$`;`
    if $\vec{I}$ `contains a solution to` $\mathcal{S}$ **then**
        u `:=` $\min(u, right(\widehat{f}(\vec{I})))$
end

These ideas can then be integrated in a branch and bound algorithm similar to the one on unconstrained optimization.[10] Our current implementation only uses simple probing techniques for inequalities and the techniques of Section 6.6 on the Fritz-John conditions for equations.

[9] Our actual implementation does a little bit better by using a naive line search when an improvement is found.

[10] When the algorithm is used in an optimistic mode (i.e., the `optimistic` keyword is included in the `Helios` statement), the upper bound is updated whenever a small box is found by the algorithm. Although the box is box-consistent, it does not mean of course that it contains a solution, which explains why the algorithm is optimistic in this case.

Benchmarks	v	d	range	Helios	HRB	CONT
Broyden	10	3^{10}	[-1,1]	1.78	18.23	
Broyden	20	3^{20}	[-1,1]	5.36	?	
Broyden	160	3^{160}	[-1,1]	95.09	?	
Broyden	160	3^{160}	$[-10^8, 10^8]$	105.610	?	
Moré-Cosnard	20	3^{20}	$[-4, 5]$	59.49	968.25	
Moré-Cosnard	40	3^{40}	$[-4, 5]$	535.70	?	
Moré-Cosnard	40	3^{40}	$[-10^8, 0]$	538.41	?	
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.

9 Experimental Results

We now turn to the experimental results of Helios on traditional benchmarks. We consider successively equation solving, unconstrained optimization, and constrained optimization.

9.1 Equation Solving

We start with experimental results of Helios on a variety of standard benchmarks for equation solving. The benchmarks were taken from papers on numerical analysis [27], interval analysis [8, 11, 25], and continuation methods [21, 28, 29, 43]. Complete details on the benchmarks can be found in the appendix. We also compare Helios with a traditional interval method using the Hansen-Segupta's operator, range testing, and branching. This method uses the same implementation technology as Helios and is denoted by HRB in the following.[11] Finally, we compare Helios with a state-of-the-art continuation method [43], denoted by CONT in the following. Note that all results given in this section were obtained by running Helios on a Sun Sparc 10 workstation to obtain all solutions. In addition, the final intervals 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

[11]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.

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).

There are a number of points that are worth mentioning. First, `Helios` does not perform any branching on `Broyden`, `Moré-Cosnard`, and interval benchmarks `i1, i2, i3` and `i5` contrary to most interval methods we know of. For these benchmarks, enforcing box-consistency is sufficient to obtain the unique solution. Note also the behaviour of `Helios` on these benchmarks. `Helios` is essentially linear in the size of the input for `Broyden` and in between linear and quadratic in the size of the input for `Moré-Cosnard`.

The last set of benchmarks indicate that `Helios` compares well with continuation methods. Consider the economics benchmark. For a given dimension n, the problem can be stated as the system

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

and the constants can be chosen at random. [43] reports times (on a DEC-5000/200) of about 1 second for $n = 4$, 6 seconds for $n = 5$, 50 seconds for $n = 6$ and 990 seconds for $n = 7$. `Helios` is substantially faster on this problem than this continuation method, since it takes about 47 seconds for $n = 7$. More importantly, the growth factor seems much lower in `Helios`. The continuation method has growths of about 8 and 20 when going from 5 to 6 and 6 to 7, while `Helios` has growths of about 6.72 and 5.68. Note also the results on the combustion and chemistry example. On the chemistry example, `Helios` exploits physical constraints (i.e., the variables must be nonnegative) that cannot be exploited by continuation methods. The last benchmark from neurophysiology is a bad example for `Helios`. `Helios` is very fast when the initial intervals are small, but its computation times increase significantly when the intervals get larger.

9.2 Unconstrained Optimization

Table 2 describes the results of `Helios` on unconstrained optimization. The benchmarks were taken mainly from [19, 12, 34, 36] and, for each of them, we give the number of variables, the range of the variables, the `CPU` time, and the number of splits. Full details on the benchmarks are given in the appendix. The experimental results once again exhibit a number of interesting facts. `Helios` is able to solve problems such as `Levy5` and `Levy6` in essentially linear time in the number of variables. `Helios` also solves the problems `Ratz25`, `Ratz27`, and `Ratz210` without splitting. These problems were used in [34] to study splitting strategies. Finally, `Helios` 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 `Helios` can be used with arbitrarily large intervals in these cases (without degrading the performance). It is interesting to note that it is possible to improve quite significantly these timings by not applying full box-consistency on all constraints. Future versions of the system will include pragmas to let users control these features.

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	$[-10, 10]$	1.04	1
Levy5	10	$[-10, 10]$	3.34	1
Levy5	20	$[-10, 10]$	11.82	1
Levy5	40	$[-10, 10]$	45.89	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 2: Summary of the Experimental Results on Unconstrained Optimization.

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 3: Summary of the Experimental Results on Constrained Optimization.

9.3 Constrained Optimization

We now turn to constrained optimization problems which are, in general, very difficult to solve. Table 3 summarizes some of our computation results on some of the toughest problems from [10]. 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. No special effort has been devoted to make constrained optimization run fast in Helios at this point, although these results are satisfactory for a global search method. Note also that there are almost no published results on the behaviour of interval methods for constrained optimization problems [16].

10 Discussion

The research described in this paper originated in our attempt to design a constraint programming language based on interval analysis. Early languages based on intervals such as BNR-Prolog [31] and CLP(BNR) [3] did not use some of the advanced techniques from interval analysis (e.g., the interval Newton method). As a consequence, we investigated how to introduce these techniques in a declarative way and proposed the concept of box-consistency which subsumes several techniques from interval analysis and artificial intelligence.[12]. This resulted in the design and implementation of Newton [41] which was used to solve many practical applications with an efficiency comparable or better to existing methods with a similar functionality.

In developing these applications, it became clear that most programs had a similar structure and that a modeling language could possibly be compiled down to Newton. Helios grew out of this observation. Helios is implemented by translating its statements into Newton programs which are then executed by the algorithms described in this paper. The translation process will be described in another paper. Its design was motivated both by AMPL and the numerous applications that were solved by Newton. The design evolves considerably to make the language simple and uniform and to ensure that Helios statements were almost similar to the mathematical descriptions of the applications.

There are many possible extensions to this work. Helios currently restricts attention to continuously differentiable functions (in the range considered) built from a set of primitive functions. However, it is not difficult to imagine a richer language in which users can state their own functions given a set of high-level constructs (e.g., case analysis). Helios could also be extended to perform a pre-processing of the initial statement. This would make it possible to obtain scaling of the system automatically. Finally, it would be of much interest to combine several local and global methods in a single system.

11 Conclusion

This paper introduces Helios, a language and a program to solve nonlinear constraint systems and global optimization problems using interval analysis. From a language standpoint, Helios is a modeling language allowing mathematical descriptions of nonlinear applications to be written

[12] A related technique was also proposed by [11]. See [40] for a comparison of the two approaches

almost as in scientific publications. As a consequence, it dramatically simplifies the solving of these applications. From an algorithm standpoint, `Helios` is based on a state-of-the-art algorithms, combining techniques from numerical analysis (i.e., interval analysis) and artificial intelligence (i.e., consistency techniques). `Helios` has been evaluated on a variety of benchmarks. On constraint-solving problems, it outperforms the interval methods we are aware of and is competitive with state-of-the-art continuation methods. On global optimization problems, `Helios` outperforms the interval algorithms we are aware of.

Acknowledgments

The branch and prune algorithm was developed jointly with Deepak Kapur and David McAllester. The notion of box-consistency was formalized jointly with Frédéric Benhamou. As always, special thanks to Baudouin Le Charlier who suggested several ways of improving the paper. 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] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, NY, 1983.

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

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

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

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

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

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

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

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

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

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

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

[13] R.B. Kearfott. INTBIS, A Portable Interval Newton/Bisection Package (Algorithm 681). *ACM Trans. Math. Software*, 16(2):152–157, 1990.

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

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

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

[17] V. Knueppel. PROFIL/BIAS-A Fast Interval Library. *Computing*, 53(3-4):323–335, 1994.

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

[19] 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.

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

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

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

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

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

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

[26] R.E. Moore and H. Ratschek. Inclusion Functions and Global Optimization II. *Mathematical Programming*, 41(3):341–356, 1988.

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

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

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

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

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

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

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

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

[35] 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.

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

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

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

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

[40] 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).

[41] P. Van Hentenryck and L. Michel. Newton: Constraint Programming over Nonlinear Constraints. *Science of Computer Programming.* To appear, 1996.

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

[43] 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.

A Description of the Benchmarks

This appendix describes the benchmarks used to evaluate `Helios`.

A.1 Equation Solving

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

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

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 [27] 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 [33].

Interval Arithmetic Benchmarks These are traditional benchmarks from interval arithmetic papers [24, 11]. Benchmark `i1` is the following set of equations

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

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

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

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

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

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

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

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

- 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 4: Coefficients for the Inverse Kinematics Example.

Kinematics Applications Application `kin1` comes from robotics and describes the inverse kinematics of an elbow manipulator [11]. It consists of a sparse system with 12 variables and the set of equations is as follows:

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

The second benchmark, denoted by `kin2`, is from [28] 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:

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

where the coefficients a_{ki} are given in table 4. In both examples, the initial intervals were given as $[-10^8, 10^8]$.

An Economics Modeling Application The following example is taken from [29]. It is a difficult economic modeling problem that can be scaled up to arbitrary dimensions. For a given dimension n, the problem can be stated as the system

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

and the constants can be chosen at random.

Combustion Application This problem is also from Morgan's book [29] 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 [21] 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 `Helios`. The application is from neurophysiology [43] 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.

A.2 Unconstrained Optimization

`Hump` is the three-hump camel function

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

`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

$$\begin{aligned} F_k &= \exp(-0.1kx_1) - \exp(-0.1kx_2) - A_k x_3 \\ A_k &= \exp(-0.1k) - \exp(-k) \end{aligned}$$

`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

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

`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 $t_i = i/5$.

A.3 Constrained Optimization

Problems `h95, h96, h97, h98` are instantiations of the generic problem described in Figure 21 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 22, 23, and 24.

min

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

subject to

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

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

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

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

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

Figure 21: The Constrained Optimization Problem for `h95, h96, h97, h98`.

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

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

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

Figure 22: 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 23: 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 24: The Constrained Optimization Problem for `h113`.