Excluding Regions Using Sobol Sequences in an Interval Branch-and-Prune Method for Nonlinear Systems*

Bartłomiej Jacek Kubica

Institute of Control and Computation Engineering,
Warsaw University of Technology, Poland

bkubica@elka.pw.edu.pl

Abstract

Traditional rejection/reduction tests used in branch-and-prune methods for nonlinear systems usually are based on various forms of the interval Newton operator and constraint propagation techniques. Hence, they are relatively costly. This paper considers an additional phase of a branch-and-prune method for the exclusion of regions not containing any solutions. Low-discrepancy sequences of Sobol are used to locate such regions, together with solving the interval tolerance problem and ε-inflation. Parallelization using the TBB library is also considered, and some numerical results are presented.

Keywords: interval computations, nonlinear systems, exclusion, Sobol sequences
AMS subject classifications: 65G40, 65Y05

1 Introduction

Let us consider the system of nonlinear equations, using the notation of [9]:

$$\begin{aligned} & f_i(x) = 0 \ , \ i = 1, \ldots, m \ , \\ & x \in \boldsymbol{x} \subseteq \mathbb{R}^n \ . \end{aligned} \tag{1}$$

We will develop an efficient interval method for locating all the solutions of such systems. We concentrate on the case of underdetermined systems (i.e., $m < n$), but the method should work also for well-determined case ($m = n$). Underdetermined systems are encountered in robotics (e.g., [7, 18]), in the stability theory of dynamical systems [19], in differential equations solving or seeking the Pareto set of a multicriteria problem [17], and in several other fields.

In [13], the author considered an interval solver of such systems and its shared-memory parallelization. In subsequent papers, several improvements have been considered, including various parallelization tools [10, 12] and using sophisticated tools and heuristics to increase the efficiency of the solver [11, 16].

*Submitted: February 21,2013; Revised: May 30, 2014; Accepted: June 25, 2014.

2 Basic Algorithm

Our solver uses interval methods based on interval arithmetic operations and interval extensions of basic functions so that the result of an operation on intervals is guaranteed to enclose the mathematically correct result set. We shall not define interval operations here, but refer the interested reader to several papers and textbooks, e.g., [6, 8, 20, 25].

Our solver is based on the "branch-and-prune" schema (see, e.g., [7]) that can be expressed by the following pseudocode:

Algorithm `IBP` ($\boldsymbol{x}^{(0)}$; `f`)
// $\boldsymbol{x}^{(0)}$ is the initial box, `f`$(\cdot)$ is the interval extension of the function $f\colon \mathbb{R}^n \to \mathbb{R}^m$
// L_{ver} is the list of boxes verified to contain a segment of the solution manifold
// L_{pos} is the list of boxes that possibly contain a segment of the solution manifold
$L = L_{ver} = L_{pos} = \emptyset$;
$\boldsymbol{x} = \boldsymbol{x}^{(0)}$;
`loop`
    process the box $\boldsymbol{x}$, using the rejection/reduction tests ;
    `if` ($\boldsymbol{x}$ does not contain solutions) `then`
        discard $\boldsymbol{x}$;
    `else if` ($\boldsymbol{x}$ is verified to contain a segment of solution manifold) `then`
        push (L_{ver}, $\boldsymbol{x}$) ;
    `else if` (tests subdivided $\boldsymbol{x}$ into $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$) `then`
        $\boldsymbol{x} = \boldsymbol{x}^{(1)}$;
        push (L, $\boldsymbol{x}^{(2)}$) ;
        `cycle loop` ;
    `else if` ($\boldsymbol{x}$ is small enough) `then`
        push (L_{pos}, $\boldsymbol{x}$) ;
    `end if` ;
    `if` ($\boldsymbol{x}$ was discarded or stored) `then`
        $\boldsymbol{x}$ = pop (L) ;
        `if` (L was empty) `then exit loop` ;
    `else`
        bisect ($\boldsymbol{x}$), obtaining $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$;
        $\boldsymbol{x} = \boldsymbol{x}^{(1)}$;
        push (L, $\boldsymbol{x}^{(2)}$) ;
    `end if` ;
`end loop`
`end IBP`

In the above pseudocode, the operations "push" and "pop" mean inserting and removing elements to/from the set (no matter in what form the set is represented, as a stack, queue, or a more sophisticated data structure).

The "rejection/reduction tests" in Algorithm `IBP` are described in [16], i.e.:

- switching between the component-wise Newton operator for larger boxes and Gauss-Seidel with inverse-midpoint preconditioner for smaller ones,
- the sophisticated heuristic to choose the bisected component (described in [16]).

One might suggest using some consistency operators too, but we do not apply them in the investigated algorithm for the following reasons:

- It is not certain when consistency methods are successful on underdetermined problems, and we focus on solving such problems.
- Component-wise Newton operator is similar to enforcing box-consistency, so it seems a proper replacement.
- We use the solver described in [16], so it seems appropriate not to change irrelevant features to investigate the influence of the initial exclusion phase.

There are several possible variants, also (see, e.g., [13, 11]), but in this report, only the heuristic form [16] is considered.

Algorithm `IBP` has been parallelized using TBB [2]. The parallelization was based on the concept of `parallel_do` (i.e., a few tasks execute some sort of a `do...while` loop). The initial box is passed to one of the tasks, and results of its bisection are distributed between tasks using the so-called "feeder". Hence, we do not have a list of boxes to consider there, i.e., we do not keep it explicitly.

We have two sets of solutions: possible solutions and verified solutions. They are represented by concurrent vectors provided by the TBB library.

Several other details of the implementation are given in [16].

3 Exclusion Regions

3.1 Motivation

An interval Newton operator is the main tool used by Algorithm `IBP`. It is powerful, yet relatively time consuming. An interval Newton operator requires computing the Jacobi matrix (or its analog, e.g., the slope matrix) of the system and often some expensive operations such as computing a preconditioner). Hence, it is beneficial to apply this operator only for boxes close to the solution manifold and to reject other boxes by some cheaper technique, e.g., using function values only and no higher-order information.

Schichl [22] considers several techniques of excluding regions from the branch-and-prune process. These techniques can be divided into two classes: removing regions containing no solutions and removing regions around a unique solution. All use first or even second order information, so they are of little use to us.

For underdetermined systems, we do not have isolated solutions, but continuous sets of solutions (see [14] – especially Section 4 – and the references therein for the discussion of various forms of the solution set). Hence, verifying the uniqueness is of little use, in general. Yet, it might be beneficial to remove infeasible regions early (and/or by a cheap procedure) so that the algorithm can concentrate on the vicinity of the solution manifold and verify segments of it. In other words, we want to find boxes, as large as possible, satisfying $f_i(x) > 0$ or $f_i(x) < 0$ for some $i \in \{1, \ldots, m\}$. We can consider it as a problem of finding the tolerable solution of a nonlinear equation:

$$f_i(x) = p \ , \tag{2}$$

where $p \in \boldsymbol{p}$, and $\boldsymbol{p} = [\varepsilon, +\infty]$ or $\boldsymbol{p} = [-\infty, -\varepsilon]$.

Linearizing the left side around a point t in the box $\boldsymbol{x}$, possibly its midpoint, we obtain

$$\boldsymbol{A} \cdot (\boldsymbol{x} - t) + \mathsf{f}_i(t) \subseteq \boldsymbol{p} \ , \tag{3}$$

and reduce the problem to the linear tolerance problem (see, e.g., [21, 23, 24, 25]), the problem of finding an inner box of the tolerable solution set to the interval system

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}\,, \tag{4}$$

where $\boldsymbol{A}$ is the Lipschitz matrix (see, e.g., [8]) of the function $f_i(\cdot)$, and

$$\boldsymbol{b} = \boldsymbol{p} - \mathsf{f}_i(t) + \boldsymbol{A}t.$$

Note that $\boldsymbol{A}$ is actually a vector (a matrix with a single row), but we retain the more general matrix notation. The tolerable solution set is defined to be the set

$$\{x \in \mathbb{R}^n \mid \forall A \in \boldsymbol{A}\ \exists b \in \boldsymbol{b}\ \ Ax = b\},$$

i.e., $\{x \in \mathbb{R}^n \mid \boldsymbol{A} \cdot x \subseteq \boldsymbol{b}\}$.

How can we find an inner box of the tolerable solution set to the system (4)? One of the classical results, due to Shary (see, e.g., [23, 24, 25]), is the following: if a point $t = (t_1, t_2, \ldots, t_n)^\top$ belongs to the tolerable solution set, then the interval vector

$$\boldsymbol{u} = t + r \cdot \boldsymbol{e} \tag{5}$$

also is included in the tolerable solution set if $\boldsymbol{e} = ([-1,1], \ldots, [-1,1])^\top$, and

$$r = \min_{k=1,\ldots,m} \left\{ \frac{\operatorname{rad} \boldsymbol{b}_k - \left| \operatorname{mid} \boldsymbol{b}_k - \sum_{j=1}^{n} \boldsymbol{a}_{kj}\, t_j \right|}{\sum_{j=1}^{n} |\boldsymbol{a}_{kj}|} \right\}. \tag{6}$$

In our case, formula (6) has to be modified. Since $\boldsymbol{b}$ has one of the bounds equal to infinity, its midpoint and radius are infinite. In the proof of Proposition 6.7.1 in [25], we can find the adequate formula,

$$r = \min_{k=1,\ldots,m} \min \left\{ \frac{\inf(\boldsymbol{b} \ominus \boldsymbol{A} \cdot t)_k}{\inf(\boldsymbol{A} \cdot \boldsymbol{e})_k}, \frac{\sup(\boldsymbol{b} \ominus \boldsymbol{A} \cdot t)_k}{\sup(\boldsymbol{A} \cdot \boldsymbol{e})_k} \right\}, \tag{7}$$

where $\ominus$ is subtraction in Kaucher arithmetic, following [9].

As stated above, the matrix $\boldsymbol{A}$ has only one row, so we can reduce formula (7) to

$$r = \min \left\{ \frac{\inf(\boldsymbol{b} \ominus \boldsymbol{A} \cdot t)}{\inf(\boldsymbol{A} \cdot \boldsymbol{e})}, \frac{\sup(\boldsymbol{b} \ominus \boldsymbol{A} \cdot t)}{\sup(\boldsymbol{A} \cdot \boldsymbol{e})} \right\}. \tag{8}$$

One of the expressions under minimization is equal to ∞, as either $\overline{b} = \infty$, or $-\underline{b} = \infty$. Also, $-\inf(\boldsymbol{A} \cdot \boldsymbol{e}) = \sup(\boldsymbol{A} \cdot \boldsymbol{e}) = |(\boldsymbol{A} \cdot \boldsymbol{e})| = \sum_{j=1}^{n} |\boldsymbol{a}_{ij}|$ holds.

Relying on these facts and providing the definition of $\boldsymbol{b}$, the other expression under minimization can be transformed, leading to the following formula for r in (5):

$$r = \mathrm{fl}_\nabla \left(\frac{|\mathsf{f}_i(t)| - \varepsilon}{\sum_{j=1}^{n} |\boldsymbol{a}_{ij}|} \right), \tag{9}$$

where fl_∇ denotes computing in floating-point arithmetic with rounding towards $-\infty$ [9].

How should we find t for the above formulae? Actually, we would like to exclude *several* regions centered around *several* points. For most systems of equations, *almost all* points of the domain lie in the interior of the set of solutions of either $\{x \in \boldsymbol{x} \mid$

$f_i(x) > 0\}$, or $\{x \in \boldsymbol{x} \mid f_i(x) < 0\}$ Clearly, this condition must be checked for each t (see the algorithm description in Subsection 3.3).

Choosing a set of points randomly seems promising. However, the author [15] investigated random point selection and described several disadvantages. Also, theoretical analysis (e.g., [5]) shows that there are sequences filling the area more uniformly (with respect to some discrepancy measure) than random ones. This suggests using quasi-random (instead of pseudo-random) sequences, specifically Sobol sequences.

3.2 Sobol Sequences

Sobol sequences (also known as LP_τ sequences) are an example of low-discrepancy sequences [5]. These are sequences designed to fill an area as uniformly as possible, yet in a completely deterministic manner. They are used in some variants of Monte Carlo methods called quasi-Monte Carlo methods. A Sobol sequence is constructed so that it is "well distributed" on the unit interval [5], i.e., this interval is filled uniformly, according to some discrepancy measure. Although the details are relatively complicated (see [5]), there are efficient generators based on the Gray code due to Antonov and Saleev. Open source implementations of this algorithm are available [4], giving the advantage of making the method deterministic, simplifying its investigation.

Generating sequences over a given box. Sobol sequences are defined over a unit interval $[0, 1]^n$, but they can simply be scaled in an affine manner to fill any other interval. The question is: do we want to generate the points over the whole search domain or over a subset of it?

Choosing points close to the boundaries as seeds of exclusion regions does not seem very promising, so we decided to test an additional variant, when we generate the points over some interval in the interior of the search domain. Specifically, we have chosen the box: $[\underline{x} + 0.1 \cdot \operatorname{wid} \boldsymbol{x}, \overline{x} - 0.1 \cdot \operatorname{wid} \boldsymbol{x}]$. Results presented in Section 5 suggest, that this choice is effective.

3.3 The Method

Our proposed algorithm for exclusion using Sobol sequences is

- generate a set of points from $\boldsymbol{x}$, using Sobol sequences;
- for each point t, compute the value of $f_i(t)$ for i chosen in a round-robin manner;
- for $\varepsilon = 10^{-4}$, if $f_i(t) \in [-\varepsilon, \varepsilon]$, the point is ignored;
- otherwise, generate an infeasible box around t using Formula (5);
- expand the infeasible boxes, using the ε-inflation procedure [8];
- remove the exclusion regions (i.e., infeasible boxes) obtained from the search domain $\boldsymbol{x}$ and perform the branch-and-prune procedure on their complement.

Different exclusion regions can be located in parallel, e.g., using the `parallel_for` concept of TBB, which is used in our code.

Infeasible boxes are removed from the search domain using the procedure from [8]:

```
complement_of (x, y, L)
    L = {};
    if ( x ∩ y == ∅ ) then
        push (L, y);
```

```
        return;
    end if
    for i = 1, ..., n do
        z = x_i ∩ y_i;
        if ( z̲ > y̲ ) then
            create a box w such that w_i = [y̲, z̲], w_j = y_j when j ≠ i;
            push (L, w);
        end if
        if ( z̄ < ȳ ) then
            create a box w such that w_i = [z̄, ȳ], w_j = y_j when j ≠ i;
            push (L, w);
        end if
    y_i = z;
    end for
end complement_of
```

To compute the complement of a set of boxes, we execute the `complement_of` procedure sequentially: we compute the complement of the first box, then for each box of the list L, we compute the complement of the second exclusion box, etc.

This is the only sequential part of our code. Some parallelization is possible here; elements of the list L can be processed concurrently. Yet, as this parallelization would be relatively fine-grained ,and the completion is computed quite efficiently (only floating-point comparisons are used in this procedure with no computations of transcendental functions), we have not parallelized this procedure.

After removing infeasible boxes from the domain, the branch-and-prune method is performed on the remaining region (represented by the set of boxes).

3.4 Analysis of the Exclusion Process

Early experiments have shown that results of our algorithm for exclusion using Sobol sequences are not deterministic. The number of processed boxes (gradient evaluations) and the computation time vary, due to the by parallelization of the exclusion process, as it affected the order in which excluded boxes have been cut from the initial box $x^{(0)}$. Let us consider the details of this phenomenon.

When we exclude a few boxes using our algorithm, the resulting set itself does not rely on the order of exclusion, but the number of boxes that enclose this set does. As a simple example, let us consider the situation presented on Figure 1: computing the complement of the union of two boxes (black) in a box. If we exclude the larger box first, we get seven resulting boxes, otherwise, we get nine boxes.

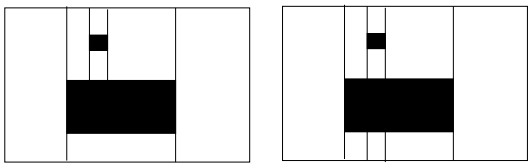

Figure 1: Subsequent exclusion of the same two boxes can result in different numbers of boxes

In a typical case, we it would seem that we should get better results when larger

boxes are removed earlier than smaller ones. This can be achieved in a simple way: after computing the exclusion regions, we sort them according to *deceasing* Lebesgue measure. The TBB library provides us a useful function `tbb::parallel_sort()` that performs this operation in a concurrent manner. Unfortunately, results of this function are not deterministic. If some boxes have identical Lebesgue measure, their order cannot be determined *a priori*, but this indeterminism seems insignificant.

4 Test Problems

The following problems used in [13] or [16] are used as numerical experiments.

The first example is a simple benchmark, two concentric circles in a plane:

$$\begin{aligned} &(x_1^2 + x_2^2 - 4) \cdot (x_1^2 + x_2^2 - 1) = 0 \ , \\ &x_1, x_2 \in [-3, 5] \ , \end{aligned} \tag{10}$$

with accuracy $\varepsilon = 10^{-5}$.

The second example is the Hippopede problem, with two equations in three variables:

$$\begin{aligned} &x_1^2 + x_2^2 - x_3 = 0 \ , \\ &x_2^2 + x_3^2 - 1.1x_3 = 0 \ . \\ &x_1 \in [-1.5, 1.5], \ x_2 \in [-1, 1], \ x_3 \in [0, 4] \ , \end{aligned} \tag{11}$$

with accuracy $\varepsilon = 10^{-7}$.

The third example, called Puma, arose in the inverse kinematics of a Puma 560 robot and is a typical benchmark for nonlinear system solvers:

$$\begin{aligned} &x_1^2 + x_2^2 - 1 = 0 \ , \ x_3^2 + x_4^2 - 1 = 0 \ , \\ &x_5^2 + x_6^2 - 1 = 0 \ , \ x_7^2 + x_8^2 - 1 = 0 \ , \\ &0.004731x_1x_3 - 0.3578x_2x_3 - 0.1238x_1 - 0.001637x_2 - 0.9338x_4 + x_7 = 0 \ , \\ &0.2238x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0 \ , \\ &x_6x_8 + 0.3578x_1 + 0.004731x_2 = 0 \ , \\ &-0.7623x_1 + 0.2238x_2 + 0.3461 = 0 \ , \\ &x_1, \ldots, x_8 \in [-1, 1] \ . \end{aligned} \tag{12}$$

In the form of (12), the Puma example is a well-determined (eight equations and eight variables) problem with 16 solutions that are easily found by several solvers. To make it underdetermined, the two last equations were dropped, as in [13]. The resulting variant with six equations was considered in numerical experiments as the third test problem, with accuracy $\varepsilon = 0.05$.

The fourth system, which we call the Rheinboldt problem, arose in aircraft equilibrium problems:

$$\begin{aligned} &-3.933x_1 + 0.107x_2 + 0.126x_3 - 9.99x_5 - 45.83x_7 - 7.64x_8 + \\ &-0.727x_2x_3 + 8.39x_3x_4 - 684.4x_4x_5 + 63.5x_4x_7 = 0 \ , \\ &-0.987x_2 - 22.95x_4 - 28.37x_6 + 0.949x_1x_3 + 0.173x_1x_5 = 0 \ , \\ &0.002x_1 - 0.235x_3 + 5.67x_5 + 0.921x_7 - 6.51x_8 - 0.716x_1x_2 + \\ &-1.578x_1x_4 + 1.132x_4x_7 = 0 \ , \end{aligned} \tag{13}$$

$$x_1 - x_4 - 0.168x_6 - x_1x_2 = 0 \ ,$$
$$-x_3 - 0.196x_5 - 0.0071x_7 + x_1x_4 = 0 \ ,$$

with accuracy $\varepsilon = 0.05$. This problem has five equations in eight variables. Since originally no bounds are given, we take $x_i \in [-2, 2]$, $i = 1, \ldots, 8$, as in [13].

The last problem is well-determined, the well-known Broyden-banded system:

$$x_i \cdot (2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j \cdot (1 + x_j) = 0, \quad i = 1, \ldots, N \ , \tag{14}$$
$$J_i = \left\{ j \mid j \neq i \text{ and } \max\{1, i-5\} \leq j \leq \min\{n, i+1\} \right\} \ ,$$
$$x_i \in [-100, 101], \quad i \in \{1, \ldots, N\} \ .$$

We consider dimensions $N = 12$ and $N = 16$.

5 Numerical Experiments

Numerical experiments were performed on a computer with 16 cores, a 8 Dual-Core AMD Opterons 8218 with 2.6GHz clock, under a Fedora 10 Linux operating system with GCC 4.6.3, glibc 2.14 and the Linux kernel 2.6.43.8.

The solver is written in C++ and compiled using the GCC compiler. The C-XSC library (version 2.5.3) [1] was used for interval computations. The parallelization (8 threads) was done with TBB 4.0, update 3 [2]. OpenBLAS 0.1 alpha 2.2 [3] was linked for BLAS operations.

We present results for the following versions of the algorithm:

- "basic" – no exclusion, only the algorithm from [16],
- "excl, N" – N exclusion regions (the number of variables of the problem), generated on the whole domain,
- "excl, $2N$" – $2 \cdot N$ exclusion regions, generated on the whole domain,
- "excl, N, inter" – N exclusion regions, generated on the box $[\underline{x} + 0.1 \cdot \operatorname{wid} \boldsymbol{x}, \overline{x} - 0.1 \cdot \operatorname{wid} \boldsymbol{x}]$,
- "excl, $2N$, inter" – as above, $2 \cdot N$ exclusion regions.

For each experiment, the following quantities are presented:

- fun. evals – number of functions evaluations,
- grad. evals – number of gradients evaluations,
- bisections,
- preconds – number of preconditioning matrix computations (i.e., performed Gauss-Seidel steps),
- bis. Newt., del. Newt. – number of boxes bisected (resp. deleted) by the interval Newton operator,
- pos.boxes, verif.boxes – number of resulting boxes in both lists – of possible and verified solutions,
- Leb.pos., Leb.verif. – Lebesgue measures of sets, covered by boxes in both lists.

Table 1: Computational results for problem (10)

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	5979	5659	5634	5496	5913
grad. evals	6708	6222	6077	6027	6442
bisections	2092	1988	1868	1896	2011
preconds	5528	5157	5075	5014	5356
bis. Newt.	39	40	34	39	38
del. Newt.	0	0	0	0	0
pos.boxes	128	128	117	118	130
verif.boxes	1061	1048	991	990	1046
Leb.pos.	2e-9	2e-9	2e-9	2e-9	3e-9
Leb.verif.	0.55	0.59	0.61	0.63	0.62
time (sec.)	< 1	< 1	< 1	< 1	< 1

Table 2: Computational results for the Hippopede problem (11)

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	1184664	560712	1121708	439598	466934
grad. evals	1361152	639616	1288326	501766	533010
bisections	329911	151299	310701	115764	125752
preconds	591814	279776	560234	219183	232866
bis. Newt.	5	15	81	14	89
del. Newt.	69760	33693	67787	24200	28118
pos.boxes	149952	63297	134961	43205	52975
verif.boxes	21672	14557	24402	16952	11400
Leb.poss.	1e-17	7e-18	1e-17	8e-18	3e-18
Leb.verif.	0.004	0.003	0.002	0.003	0.003
time (sec.)	< 1	< 1	< 1	< 1	< 1

6 Analysis of the Results

For all of the test problems, we achieved some speedup, but the improvements happen to be very "capricious," varying from minor to dramatic.

Results for underdetermined problems sometimes are hard to analyze, as we might have a tradeoff between computation time and precision of enclosing the solution manifold. Thankfully, this did not occur in our experiments. Only in Table 4 ,one version of the method (specifically, "$2N$ excl") results in a relatively crude approximation of guaranteed boxes, yet computation time is not saved by that.

We tried to investigate the number of exclusion regions. In [15], we claimed that the number of regions equal to the number of variables was optimal. Indeed, for most problems, the version generating N points from the internal subinterval of the domain

Table 3: Computational results for the Puma problem (12), with 6 equations

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	3673215	3442659	3376097	3621021	3444777
grad. evals	3236880	2978364	2940528	3163032	3039000
bisections	269711	247685	244245	263205	251761
preconds	461464	417272	412512	447832	429851
bis. Newt.	0	78	104	99	197
del. Newt.	54280	49752	45731	53495	47747
pos.boxes	188208	173440	174344	184904	180288
verif.boxes	3744	2600	2192	2520	2168
Leb.poss.	2e-9	3e-9	2e-9	3e-9	3e-9
Leb.verif.	4e-11	5e-8	2e-7	3e-7	3e-9
time (sec.)	4	4	4	4	4

Table 4: Computational results for the Rheinboldt problem (13)

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	213645211	186210881	183873684	180742165	180380236
grad. evals	128791915	112809925	111877105	107399315	109435240
bisections	12225817	10688351	10604774	10116277	10381309
preconds	21095388	18622946	18525362	17586376	18084865
bis. Newt.	4401	5057	4880	5106	4890
del. Newt.	3293093	2835923	2846290	2724531	2766737
pos.boxes	7684286	6828040	6759768	6422468	6639564
verif.boxes	486738	425256	423483	404854	398901
Leb.poss.	1e-6	1e-6	1e-6	1e-6	1e-6
Leb.verif.	0.003	0.070	4.091	0.052	0.040
time (sec.)	232	202	198	188	192

resulted in the lowest computation time (and gradient evaluations). Yet for some problems, such as the Broyden-banded system with $N = 16$ (Table 6), a higher number (e.g., twice the number of variables) performs better. Nevertheless, the difference was not tremendous, and for some problems, using the larger number of exclusion regions performed significantly worse, for example, Broyden-banded with $N = 12$ (Table 5) or the Hippopede problem (Table 2).

For Table (1), all algorithm versions performed in a similar way, but the version with N internal points results also in the smallest number of gradient evaluations.

It seems clear that generating points on some subinterval of the interior of the search domain outperformed generating them over the whole search domain.

Table 5: Computational results for the Broyden-banded problem (14), $N = 12$

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	23364196	19432059	19173335	14561810	32407927
grad. evals	8625492	6722376	6617988	5499084	10143900
bisections	337884	264036	254267	213554	397373
preconds	138663	77903	88259	93771	107581
bis. Newt.	21510	14642	14620	14071	17614
del. Newt.	205614	173386	168008	127934	282202
pos.boxes	0	0	0	0	0
verif.boxes	1	1	1	1	1
Leb.poss.	0.0	0.0	0.0	0.0	0.0
Leb.verif.	2e-114	1e-117	3e-97	9e-105	4e-105
time (sec.)	21	16	16	14	25

Table 6: Computational results for the Broyden-banded problem (14), $N = 16$

version	basic	excl, N	excl, $2N$	excl, N, inter	excl, $2N$, inter
fun. evals	7975494792	4705422366	4564484468	2892322332	2771968224
grad. evals	2139405184	1257731440	1193654000	834621936	810178304
bisections	66082093	38905745	36790866	25747517	24752411
preconds	24741064	9628066	10304221	8552837	6374714
bis. Newt.	774318	394943	476259	331213	525063
del. Newt.	50634661	28798049	28148442	18288848	17249029
pos.boxes	0	0	0	0	0
verif.boxes	1	1	1	1	1
Leb.poss.	0.0	0.0	0.0	0.0	0.0
Leb.verif.	3e-119	8e-129	1e-118	4e-137	3e-186
time (sec.)	6911	4035	3826	2597	2485

7 Conclusions

We have presented an improved version of a previously developed parallel interval solver for (underdetermined) nonlinear systems. The improvement is based on an initial procedure to delete regions not containing solutions. This procedure uses Sobol low-discrepancy sequences, interval linear tolerance problem theory, and ε-inflation.

Overall, the method performs quite well (even if not tuned in the optimal way), and it parallelizes simply using the Intel TBB library.

Acknowledgments

The author would like to acknowledge many fruitful and inspiring discussions with colleague Adam Woźniak. The author is grateful to an anonymous referee for helping to make the paper more clear.

References

[1] C-XSC interval library. `http://www.xsc.de`.

[2] Intel TBB. `http://www.threadingbuildingblocks.org`.

[3] OpenBLAS library. `http://xianyi.github.com/OpenBLAS/`.

[4] Sobol sequence generator. `http://web.maths.unsw.edu.au/~fkuo/sobol/`.

[5] Michael Drmota and Robert F. Tichy. *Sequences, Discrepancies and Applications*. Springer, 1997.

[6] Eldon Hansen and William Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 2004.

[7] Daisuke Ishii, Alexandre Goldsztejn, and Christophe Jermann. Interval-based projection method for under-constrained numerical systems. *Constraints*, 17 (4):432–460, 2012.

[8] Ralph B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer, Dordrecht, 1996.

[9] Ralph B. Kearfott, Mitsuhiro T. Nakao, Arnold Neumaier, Siegfried M. Rump, Sergey P. Shary, and Pascal van Hentenryck. Standardized notation in interval analysis. *Vychislennyie Tekhnologii (Computational Technologies)*, 15(1):7–13, 2010.

[10] Bartłomiej J. Kubica. Intel TBB as a tool for parallelization of an interval solver of nonlinear equations systems. Technical Report 09-02, Institute of Control and Computation Engineering, Warsaw University of Technology, 2009.

[11] Bartłomiej J. Kubica. Performance inversion of interval Newton narrowing operators. *Prace Naukowe Politechniki Warszawskiej. Elektronika*, 169:111–119, 2009. KAEiOG 2009 Proceedings.

[12] Bartłomiej J. Kubica. Shared-memory parallelization of an interval equations systems solver – comparison of tools. *Prace Naukowe Politechniki Warszawskiej. Elektronika*, 169:121–128, 2009. KAEiOG 2009 Proceedings.

[13] Bartłomiej J. Kubica. Interval methods for solving underdetermined nonlinear equations systems. *Reliable Computing*, 15:207–217, 2011. SCAN 2008 Proceedings.

[14] Bartłomiej J. Kubica. A class of problems that can be solved using interval algorithms. *Computing*, 94:271–280, 2012. SCAN 2010 Proceedings.

[15] Bartłomiej J. Kubica. Exclusion regions in the interval solver of underdetermined nonlinear systems. Technical Report 12-01, Institute of Control and Computation Engineering, Warsaw University of Technology, 2012.

[16] Bartłomiej J. Kubica. Tuning the multithreaded interval method for solving underdetermined systems of nonlinear equations. *Lecture Notes on Computer Sciense*, 7204:467–476, 2012. PPAM 2011 Proceedings.

[17] Bartłomiej J. Kubica and Adam Woźniak. Using the second-order information in Pareto-set computations of a multi-criteria problem. *Lecture Notes on Computer Scienses*, 7134:137–147, 2012. PARA 2010 Proceedings.

[18] Jean-Pierre Merlet. Interval analysis for certified numerical solution of problems in robotics. *International Journal of Applied Mathematics and Computer Science*, 19 (3):399–412, 2009.

[19] Arnold Neumaier. The enclosure of solutions of parameter-dependent systems of equations. In *Reliability in Computing*, pages 269–286. Academic Press, 1988.

[20] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[21] Jiri Rohn. Inner solutions of linear interval systems. In *Interval Mathematics*, pages 157–158. Springer-Verlag, New York, 1986.

[22] Herman Schichl and Arnold Neumaier. Exclusion regions for systems of equations. *SIAM Journal of Numerical Analysis*, 42:383–408, 2004.

[23] Irene A. Sharaya. The largest interval of given proportions for the interval linear tolerable solution set. `http://conf.nsc.ru/files/conferences/niknik-90/fulltext/38714/49309/Sharaya.pdf`, 2011. NIKNIK-90 International Conference.

[24] Sergey P. Shary. An interval linear tolerance problem. *Automation and Remote Control*, 65:1653–1666, 2004.

[25] Sergey P. Shary. *Finite-dimensional Interval Analysis*. XYZ, Novosibirsk, 2013. (in Russian) Electronic book, `http://www.nsc.ru/interval/Library/InteBooks/SharyBook.pdf` (accessed 2014.05.15).