Numer Algor (2015) 70:929–963
DOI 10.1007/s11075-015-9980-y

ORIGINAL PAPER

Presentation of a highly tuned multithreaded interval solver for underdetermined and well-determined nonlinear systems

Empirical evaluation of innovations

Bartłomiej Jacek Kubica

Received: 1 September 2014 / Accepted: 16 February 2015 / Published online: 24 March 2015

Abstract The paper summarizes author's investigations in tuning a multithreaded interval branch-and-prune algorithm for nonlinear systems and presents the developed solver. New results for using the box-consistency enforcing operator and a new variant of the initial exclusion phase are presented. Also, a new heuristic to choose the coordinate for bisection is considered. Extensive numerical experiments are analyzed to provide the satisfying version of the algorithm.

Keywords Interval methods · Nonlinear systems of equations · Heuristics · Low-discrepancy sequences · Multithreaded computations

1 Introduction

We consider the problem of finding *all* solutions of nonlinear systems of equations, i.e., systems of the form:

$$\begin{aligned} f(x) &= 0\,, \\ x &\in [\underline{x}, \overline{x}]\,, \end{aligned} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^m$, $m \leq n$.

Such systems are ubiquitous in several branches of science and engineering. Many of them are not well-determined, but underdetermined, i.e., having fewer equations than unknowns ($m < n$), which means they have uncountably many solutions and their solution sets do not consist of isolated points, but are manifolds. In particular,

B. J. Kubica (✉)
Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: bkubica@elka.pw.edu.pl

we encounter such systems in robotics [19], stability theory of dynamical systems [35], differential equations solving [31] and multicriteria analysis [30].

Example As a specific example, we can consider solving the inverse kinematic problem of a serial planar nR-manipulator, i.e., a manipulator working in the XOY space and consisting of n rotational joints. Assume, the kinematic chain starts in the point (0, 0) and the effector is supposed to be placed in the point (1, 1) and oriented orthogonally (under the right angle) to the OY axis. This problem can be formulated as the following system of equations:

$$\sum_{i=1}^{n} l_i \cdot \prod_{j=1}^{i} \cos\left(\sum_{k=1}^{j} x_k\right) - 1 = 0\,,$$

$$\sum_{i=1}^{n} l_i \cdot \prod_{j=1}^{i} \sin\left(\sum_{k=1}^{j} x_k\right) - 1 = 0\,,$$

$$\sum_{i=1}^{n} x_i - \frac{\pi}{2} = 0\,,$$

$$x_i \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \quad i = 1, \ldots, n\,.$$

We assume $l_i = 1.0$ for $i = 1, \ldots, n$.

For $n = 3$ the problem is well determined – there are exactly two manipulator configurations satisfying the constraints (see Fig. 1, on the left). But for $n = 5$, the set of possible manipulator configurations is a manifold – it is of the measure continuum. A few example configurations are presented on the right part of Fig. 1

Interval methods (see, e.g., [18, 20, 37]) are a well-known approach to find all solutions of both kinds of systems. Their essence is to perform operations on (possibly multidimensional) intervals (so-called *boxes* in $\mathbb{R}^n$; see Fig. 2) instead of specific numbers (vectors), so that, if $a \in \mathbf{a}$ and $b \in \mathbf{b}$, then ($a \odot b \in \mathbf{a} \odot \mathbf{b}$), i.e., the result of an operation on numbers belongs to the result of operation on intervals, containing the arguments. This leads to interval arithmetic operations and definitions of basic functions operating on intervals. We shall not define basic interval operations here; the interested reader is referred to several papers and textbooks, e.g., [18, 20, 37].

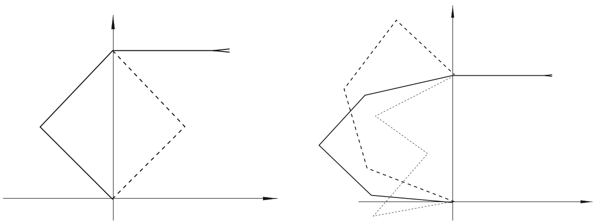

Fig. 1 Left: both feasible 3R manipulator configurations, right: three examples of uncountably many feasible 5R manipulator configurations

In the previous series of papers ([22–29]) the author presented an interval solver for such systems and investigated several acceleration tools. The solver is targeted at underdetermined problems, yet it could be used for well-determined ones, also.

2 Generic algorithm

The solver is based on the branch-and-prune (B&P) schema that can be expressed by pseudocode presented in Algorithm 1.

Algorithm 1 IBP

Require: L, f, ε

{L is the list of initial boxes – often containing a single box $\mathbf{x}^{(0)}$}
{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}
$L_{ver} = L_{pos} = \emptyset$
$\mathbf{x}$ = pop (L)
loop
  process the box $\mathbf{x}$, using the rejection/reduction tests
  if ($\mathbf{x}$ does not contain solutions) **then**
    discard $\mathbf{x}$
  else if ($\mathbf{x}$ is verified to contain a segment of the solution manifold) **then**
    push (L_{ver}, $\mathbf{x}$)
  else if (the tests resulted in two subboxes of $\mathbf{x}$: $\mathbf{x}^{(1)}$ and$\mathbf{x}^{(2)}$) **then**
    $\mathbf{x} = \mathbf{x}^{(1)}$
    push (L, $\mathbf{x}^{(2)}$)
    cycle loop
  else if (wid $\mathbf{x} < \varepsilon$) **then**
    {The box $\mathbf{x}$ is too small for bisection}
    push (L_{pos}, $\mathbf{x}$)
  end if
  if ($\mathbf{x}$ was discarded **or** $\mathbf{x}$ was stored) **then**
$\mathbf{x}$ = pop (L)
  if (L was empty) **then**
    {all boxes have been considered}
    return L_{ver}, L_{pos}
  end if
else
    bisect ($\mathbf{x}$), obtaining $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$
  $\mathbf{x} = \mathbf{x}^{(1)}$
  push (L, $\mathbf{x}^{(2)}$)
  end if
end loop

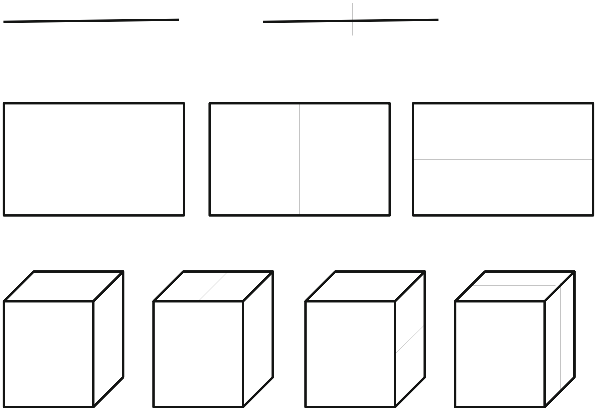

Fig. 2 Bisection of an interval and two- and three-dimensional boxes

Operations “push” and “pop” in the algorithm, mean inserting and removing elements to/from the set (the names will be used independently on how the set is represented – as a stack, queue or a more sophisticated data structure).

The precision parameter ε can have various values. Usually, 10^{-7}-10^{-6} are sufficient values, but for hard problems (especially underdetermined ones), we have to content ourselves with larger thresholds; or the computation will take too much time.

The bisection operation (or – to be more general – subdivision of a box) slices a box into subboxes. Usually, one of the edges of the box is splitted in the midpoint and that is the approach we use (see Fig. 2).

Algorithm 1 allows to find all solutions of the problem, yet it can be time-consuming and memory-demanding. Because of this, it is very important to choose proper “rejection/reduction tests” (mentioned in Algorithm 1) to tune the efficiency as much as possible. Fortunately, the algorithm can be parallelized (see, e.g., [24]), as processing different boxes can be performed independently. Obviously, the lists L, L_{ver} and L_{pos} have to be implemented in a multithreaded-safe way; so do other used tools.

The “rejection/reduction tests”, mentioned above may vary. Several of them are described in previous papers of the author, specifically [27–29], i.e.:

– various kinds of the interval Newton operator and switching between the componentwise Newton operator (for larger boxes) and Gauss-Seidel with inverse-midpoint preconditioner, for smaller ones,
– a sophisticated heuristic to choose the bisected component [27],
– an initial exclusion phase of the algorithm (deleting some regions, not containing solutions) – based on Sobol sequences [28],
– an additional test based on quadratic approximation of a single equation and the Hansen’s method [18] to solve quadratic equations with interval coefficients; see [29].

There are many other tools, also. Some of them are not suitable for multi-threaded computations as they use, e.g., linear programming while popular linear programming solvers are either inefficient (e.g., the solver used in the C-XSC library [1]) or not MT-safe, e.g., the solver GLPK [5]. Hence, we do not consider some popular tools, like LP-preconditioners of [20] or LP-narrowing.

As Algorithm 1 is, in general, time-consuming and memory-demanding, it is crucial to provide a proper heuristic to choose and parameterize the rejection/reduction tests efficient for a specific class of problems.

In mentioned papers, the author considered several tools and proposed some policies to apply them. Yet, as there are so many of these tools, specific cooperation between them and tuning of the heuristics, remains to be determined.

3 Box consistency enforcing

One of the tools to improve the performance of Algorithm 1 are so-called *consistency operators*. They have not been considered in previous papers of the author. As reported, e.g., in [18], enforcing some partial consistencies can be very efficient on large boxes.

There are several kinds of partial consistencies, the most commonly used being box-consistency (BC), described, e.g., by [14] and hull-consistency (HC) – see, e.g., [10]. The latter requires complicated decomposition of the expression into a syntactic tree, so we decided not to use it in the current version of our method (unless we consider the quadratic approximation of [29] a very specific instance of HC). Hence, box consistency can be enforced using the unidimensional Newton operator that is easy to implement.

The idea of box consistency is to find the leftmost and rightmost "pseudo-solutions" of a constraint [12], i.e., intervals $\left[x_i^*, x_i^{*+}\right]$, such that:

$$0 \in \mathsf{f}\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{i-1}, \left[x_i^*, x_i^{*+}\right], \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n\right) \ , \tag{2}$$

where the interval x_i^{*+} is the next representable floating-point number, after x_i^*, i.e., $\left[x_i^*, x_i^{*+}\right]$ is the smallest representable interval (such intervals are called *canonical* intervals).

Formula (2) is valid for equations, but it can be adapted for inequalities and other types of constraints, also.

The algorithm, we use is usually called BC3; as an analog of AC3 (AC stands for "arc consistency"; see, e.g., [11]). Usually, the algorithm is formulated as subsequent calls to recursive procedures "left_narrow" and "right_narrow", computing leftmost and rightmost pseudo-solutions. As specific implementations vary in several details (see, e.g., [10, 12, 14, 15]), the author presents pseudocodes for his own implementation. Procedure "left_narrow" is described by Algorithm 2 ("right_narrow" is analogous) and the overall BC3 procedure – by Algorithm 3.

Please note, we break the procedure if the progress is not sufficient. The parameter ε_{equal} is used for this purpose and the proper condition is checked in line 10 of Algorithm 3.

Algorithm 2 Procedure left_narrow

Require: $\mathbf{x}$, f, i, ε, ε_{equal}
$x_{left} = [\underline{x}_i, \underline{x}_i^+]$
if $(0 \in \mathsf{f}(x_{left}))$ **then**
  return $\underline{x}_{left}$, "found a pseudo-solution"
end if
compute the interval extension of $\mathbf{g}_i = \frac{\partial f}{\partial x_i}(\mathbf{x})$
{Using the automatic differentiation arithmetic makes us compute the whole interval gradient $\mathbf{g}$, but only one component is going to be used}
update $\underline{x} = \overline{x}_{left}$
compute $\mathbf{x}_{new} = \mathbf{x}_{left} - \frac{\mathsf{f}(\mathbf{x}_{left})}{\mathbf{g}_i}$ {using ordinary or extended interval arithmetic}
if $(\mathbf{x}_i \cap \mathbf{x}_{new} = \emptyset)$ **then**
  return "no solution"
end if
if $(\mathrm{dist}(\mathbf{x}_i, \mathbf{x}_{new}) < \varepsilon_{equal})$ **then**
update $\mathbf{x}_i = \mathbf{x}_i \cap \mathbf{x}_{new}$
  return $\underline{x}_i$, "found a pseudo-solution"
end if
update $\mathbf{x}_i = \mathbf{x}_i \cap \mathbf{x}_{new}$
if $\{(\mathrm{wid}\, \mathbf{x}_i \leq \varepsilon)\}$ **then**
  {The component $\mathbf{x}_i$ too narrow for bisection}
  return $\underline{x}_i$, "found a pseudo-solution"
end if
bisect $\mathbf{x}_i$, obtaining $\mathbf{x}_i^{(1)}$ and $\mathbf{x}_i^{(2)}$
if (left_narrow $(\mathbf{x}^{(1)}$, f, i, $\varepsilon)$ results in (x^*, "found a pseudo-solution")) **then**
  return x^*, "found a pseudo-solution"
end if
if (left_narrow $(\mathbf{x}^{(2)}$, f, i, $\varepsilon)$ results in (x^*, "found a pseudo-solution")) **then**
  return x^*, "found a pseudo-solution"
end if
return "no solution"

The parameter ε_{equal} is used as the threshold value for braking the BC3 procedure – we set $\varepsilon_{equal} = 10^{-4}$. Such policy is not used in other known versions of the BC3 procedure; yet, it performs well in our algorithm and seems to improve the efficiency (results proving it are not presented due to the lack of space).

Heuristic But for which boxes should we apply the BC3 procedure? For "sufficiently large", as pointed above. The heuristic we propose is described by Algorithm 4. It suffices that a single edge of the box is longer than the threshold value ε_{bc3}. In Section 6 we consider two possible values of ε_{bc3}: $\frac{3}{n}$ and $\frac{6}{n}$; results for $\frac{1.5}{n}$ are not presented as they occurred to be less promising. As for ε_{Ncmp}, the value $\frac{1}{n}$ was used – twice larger than in [27] to emphasize the importance of the componentwise operator, partially replaced by the BC3 procedure.

Algorithm 3 Procedure bc3revise

Require: $\mathbf{x}$, f, i, ε, ε_{equal}
repeat
  store $\mathbf{x}^{old} = \mathbf{x}$
  modified = **false**
  if (left_narrow ($\mathbf{x}$, f, i, ε) results in “no solutions”) **then**
    return “no solutions”
  end if
  if (right_narrow ($\mathbf{x}$, f, i, ε) results in “no solutions”) **then**
    return “no solutions”
  end if
    if ($\text{dist}(\mathbf{x}_i, \mathbf{x}_i^{old}) >= \varepsilon_{equal}$) **then**
    modified = **true**
  end if
until (not modified)
return x

Operators Ncmp and GS are described, e.g., in [23, 25, 27]. For the sake of brevity, formulae are as follows:

$$N_{cmp}(\mathbf{x}, \check{x}, \mathsf{f}, i, j) = \check{x}_j - \frac{\mathsf{f}_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}, \check{x}_j, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_n)}{\frac{\partial \mathsf{f}_i}{\partial x_j}(\mathbf{x}_1, \ldots, \mathbf{x}_n)},$$

$$GS(\mathbf{x}, \check{x}, \mathsf{f}, i) = \check{x}_i - \Bigg(Y_{i:} \cdot \mathsf{f}(\check{x}_1, \ldots, \check{x}_m, \mathbf{x}_{m+1}, \ldots, \mathbf{x}_n) + \\ + \sum_{j=1, j\neq i}^{m} Y_{i:} \cdot \mathbf{J}_{:j} \cdot (\mathbf{x}_j - \check{x}_j) \Bigg) / (Y_{i:} \cdot \mathbf{J}_{:i}) \, .$$

The quantity $\check{x}$, in the above formulae, is chosen to be mid $\mathbf{x}$.

Algorithm 4 Heuristic-BC3

Require: $\mathbf{x}$, f, ε_{bc3} **then**
if ($\exists i$ wid $(\mathbf{x})_i > \varepsilon_{bc3}$) **then**
  perform bc3revise procedure
end if
if (there are at least $(n - m)$ components of $\mathbf{x}$, for which wid $(\mathbf{x}_i) > \varepsilon_{Ncmp}$) **then**
  use the Ncmp operator
 else
  use the GS operator
end if

4 Initial exclusion phase

This tool, proposed by the author, has been described in [26, 28]. Before starting the B&P method (i.e., Algorithm 1), we perform the procedure described by Algorithm 5.

Algorithm 5 The initial exclusion phase

Step 1. Generate N_{Sobol} points, covering the unit box $[0, 1]^n$. Generate N_{Sobol} points $t^{(i)}$, covering the domain $x^{(0)}$, by affine transformation of points, generated in Step 1.
Step 2. Let $L_{excl} = \emptyset$ be the list of boxes to exclude.
Step 3. For each $i = 1, \ldots, N_{Sobol}$ perform the following steps.
Step 4. Choose the equation number j and compute $f_j(t^{(i)})$.
Step 5. If $f_j(t^{(i)}) \in [-\varepsilon, \varepsilon]$, increment i and – if $i <= N_{sobol}$ – return to Step 4, else go to Step 10.
Step 6. Generate the infeasible box $\mathbf{x}^{(i)}_{excl}$ around $t^{(i)}$, using the approach of Shary [36].
Step 7. Expand $\mathbf{x}^{(i)}_{excl}$, using the ε-inflation procedure; see, e.g., [20].
Step 8. Store $\mathbf{x}^{(i)}_{excl}$ in the list L_{excl}.
Step 9. Increment i and – if $i <= N_{sobol}$ – return to Step 4.
Step 10. Compute the complement of box-set L_{excl}.

Algorithm 5 does not specify several important features:

– how to compute the elements of sequence $(t^{(i)})$,
– which equation number j to use for the point $t^{(i)}$; in [28] we used a round-robin policy (the equation number $j = i \mod m$ was used), which was quite an arbitrary assignment; an alternative is presented in one of the below paragraphs,
– how to compute the complement of the created box-set L_{excl} in $x^{(0)}$.

Computing the Sobol sequence is a relatively complex task, but there exist efficient and well-know algorithms (based on Gray code) and even open-source implementations (e.g., [8]). Surprisingly, a more difficult problem is computing the complement of the set of excluded boxes.

Using all equations for the exclusion Instead of arbitrary choosing a single equation for each $(t^{(i)})$ in Algorithm 5, we can use all of the equations. Modification of the code is very simple and does not even require an additional loop.

For each point we start with the equation $f_j(x) = 0$ for $j = 1$. After realizing – in Step 7 – that we can no longer expand the box for this equation, we increment j and proceed. The ε-inflation procedure is broken when j becomes m and no progress can be obtained for the last equation. Eventually, we choose j for which the expanded box had the largest Lebesgue measure.

Additionally, we can expand the box even further if the problem is sparse. For each j the excluded box, all variables k such that $\frac{\partial f_j}{\partial x_k} = 0$ in the whole domain, can be set to $\mathbf{x}_k = \mathbf{x}^{(0)}_k$. Let us call this technique *sparsity-based expanding*.

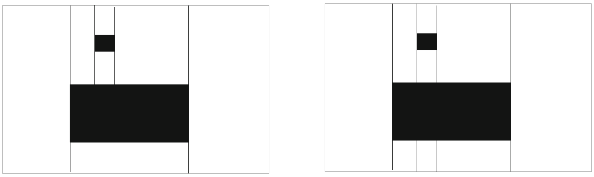

Fig. 3 Result of exclusion of two boxes – the larger or the smaller box is excluded first

Complement of a box-set There exist a well-known procedure – described by [20] – to compute the complement of a single box; we present it in Algorithm 6. The complement of the box-set can be computed by Algorithm 7, yet the procedure is inefficient. Not only, it does not parallelize well, but the resulting box-set may consist of too many boxes – see Fig. 3 (also described in [28]).

Algorithm 6 Complement of the box $\mathbf{x}^{excl}$ in $\mathbf{x}$

Require: $\mathbf{x}^{excl}$, $\mathbf{x}$, L

$L = \{\}$
if ($\mathbf{x}^{excl} \cap \mathbf{x} = \emptyset$) **then**
    push (L, $\mathbf{x}$)
    return
end if
for ($i = 1, \ldots, n$) **do**
    $\mathbf{z} = \mathbf{x}_i^{excl} \cap \mathbf{x}_i$
    if ($\underline{z} > \underline{x}_i$)
      create a box $\mathbf{w}$ such that $\mathbf{w}_i = [\underline{x}_i, \underline{z}]$, $\mathbf{w}_j = \mathbf{x}_j$ when $j \neq i$
      push (L, $\mathbf{w}$)
    end if
    if ($\overline{z} < \overline{x}_i$) **then**
      create a box $\mathbf{w}$ such that $\mathbf{w}_i = [\overline{z}, \overline{x}_i]$, $\mathbf{w}_j = \mathbf{x}_j$ when $j \neq i$
      push (L, $\mathbf{w}$)
    end if
    $\mathbf{x}_i = \mathbf{z}$
end for
return L

The problem is that we exclude the larger $\mathbf{x}^{excl}$ box from $\mathbf{x}$ first, but the larger excluded box can have a smaller intersection ($\mathbf{x}^{excl} \cap \mathbf{x}$) with $\mathbf{x}$.

The simple improvement is to exclude from each box $\mathbf{x}$ the box that has the largest intersection with it – so we exclude different boxes from different parts of the domain at the same time, probably. Such a procedure can be parallelized, simply – using the task-parallelism model, which is used in TBB [2]. The procedure is described by Algorithm 8

Algorithm 7 Old-exclusion-procedure

Require: $\mathbf{x}^{(0)}$, L_{excl}
sort L_{excl} with respect to decreasing Lebesgue measure
$L1 = \{\mathbf{x}^{(0)}\}$
for all $\mathbf{x}^{excl} \in L_{excl}$ **do**
  compute the complement of $\mathbf{x}^{excl}$ in $L1$ and store in $L2$
  $L1 = L2$
end for
return $L1$

Yet another feature, used in the ultimate version of the algorithm, is not to exclude all boxes from L_{excl}. When we obtain the given number of boxes in L – $N_{cutoff} = 128$ occurred to be a good choice – now boxes are not excluded, but $\mathbf{x}$'s from the remaining tasks are inserted into L directly. This trick might seem peculiar, but it improves the performance, significantly.

Remark TBB templates `tbb::parallel_do` and `tbb::parallel_do_feeder` are very suitable for the implementation. The former allows a concurrent execution of a `do...while` loop, i.e., executing the same procedure for an unknown number (unlike `parallel_for`) of arguments. In our case: concurrent executions of Algorithm 8. And adding additional tasks is performed by a dedicated "feeder" object. Details can be found in [22] or, directly in TBB documentation [2].

Algorithm 8 New-exclusion-procedure

Require: task ($\mathbf{x}$, L_{excl})
{Obviously, we start with the task ($\mathbf{x}^{(0)}$, L_{excl}).}
{All tasks put the boxes (with synchronization) to the list L of Algorithm 1.}
choose $\mathbf{x}^{excl}$ from L_{excl}, such that the Lebesgue measure of $\mathbf{x} \cap \mathbf{x}^{excl}$ is maximized
  if (this measure is lower than ε) **then**
    {It is not beneficial to compute the complement of these boxes.}
    return
end if
remove $\mathbf{x}^{excl}$ from L_{excl}
compute the complement of $\mathbf{x}^{excl}$ in $\mathbf{x}$ and store in L_{task}
if ($L == \{\}$) **then**
  for all $\mathbf{x}^{new} \in L_{task}$ **do**
    push (L, $\mathbf{x}^{new}$)
  end for
  return
end if
for all $\mathbf{x}^{new} \in L_{task}$ **do**
  create task (x^{new}, L_{excl})
end for

5 Choosing the coordinate for bisection

In [27] the problem of choosing the proper variable for bisection has been discussed. We emphasized the insufficiency of earlier approaches (see, e.g., [9]) and proposed the heuristic, described by Algorithm 9.

Its main idea was not to bisect the component that is the longest or has the maximal smear, but the one *that will cause the resulting boxes to be convenient for the Newton operator to narrow*. This led to the idea of choosing the component with the *minimal magnitude*. On the other hand, bisecting such components only, would result in loosing the convergence (also, it is not beneficial to have large differences between the component length, so if the difference between the longest and shortest component is too large, it is good to bisect the longest component). Hence we obtain a relatively complicated policy, trying to take into account all these facts. It is described by Algorithm 9.

Algorithm 9 Choosing the variable for bisection of $\mathbf{x}$ – heuristic from [27]

Require: $\mathbf{x}$ {We assume the procedure gets sufficient info about the results of the Newton operator evaluation, also – see below}
1: FindMaxDiam($\mathbf{x}$, j_{max}, w_{max})
2: FindMinDiam($\mathbf{x}$, j_{min}, w_{min})
3: FindMaxDiamUnnarrowed($\mathbf{x}$, $j_{max\,unn}$, $w_{max\,unn}$) {Find the index and diameter of the longest component *not reduced* by the last use of the Newton operator}
4: **if** (Newton reduced no components or $w_{max} > 1.5 \cdot w_{max\,unn}$) **then**
5:     **return** j_{max}
6: **else if** ($w_{max\,unn} > 8 \cdot w_{min}$) **then**
7:     **return** $j_{max\,unn}$
8: **end if**
9: FindSmallestMaxMag($\mathbf{x}$, j, w) {Find the component with the smallest maximal magnitude of the Jacobi matrix in all rows}
10: **if** ($w > 0.1$) **then**
11:     **return** j
12: **else**
13:     **return** $j_{max\,unn}$
14: **end if**

The algorithm was designed for underdetermined problems, but experiments in [27] have shown some improvements for well-determined problems, also.

A careful analysis shows, that the main reason of this improvement is avoiding to choose the components, narrowed by the Newton operator (by a narrowed component, we mean the one for which the operator had improved both bounds, i.e., $\mathbf{x}_i^{new} \subset \operatorname{int} \mathbf{x}_i$).

Should we choose the minimal magnitude components, indeed? For well-determined problems, it is not beneficial, certainly – we should bisect components with the maximal magnitude as they have the largest influence on the overestimation of the solved functions. For underdetermined problems, the situation is more complicated. The above argument holds, but the component with the maximal magnitude is the one that should be narrowed by the Newton operator (for underdetermined problems *not all* components are narrowed to verify the solution existence!). Experiments with the MaxSumMag and MaxSmear heuristics (most of them are not presented due to lack of space, see also [27]) show a very poor performance of such policies for underdetermined problems.

Consequently, we propose to stick to choosing the maximal diameter for boxes that are not narrowed yet. For boxes where some components have already been narrowed, we can use the maximal sum magnitude heuristic, but only on *unnarrowed components*. It occurred that for smaller boxes, it is better to switch to the maximal diameter again (but, also, not bisecting the narrowed components).

For well-determined problems, the MaxSumMagnitude performs well, in general, but an exception to it is the Brent10 problem. Hence, we switch to MaxDiamUnnarrowed on occasions.

Details are given by the pseudocode in Algorithm 10.

Algorithm 10 Choosing the variable for bisection of $\mathbf{x}$ – the new heuristic

Require: $\mathbf{x}$ {We assume the procedure gets sufficient info about the results of the Newton operator evaluation, also – see below}

FindMaxDiamUnnarrowed($\mathbf{x}$, $j_{max\,unn}$, $w_{max\,unn}$)
FindMaxSumMagnitudeUnnarrowed($\mathbf{x}$, $j_{max\,mag}$, $w_{max\,mag}$)
if (Newton reduced no components) **then**
  if ($m < n$ (i.e., the problem is underdetermined) **or** $w_{max\,unn} \geq 16 \cdot w_{max\,mag}$) **then**
    return $j_{max\,unn}$
  else
    return $j_{max\,mag}$
  end if
else
  if ($w_{max\,mag} \geq 0.1$) **then**
    return $j_{max\,mag}$
  else
    return $j_{max\,unn}$
  end if
end if

6 Computational experiments

Numerical experiments were performed on a computer with 4 cores (allowing hyperthreading), i.e., an Intel Core i7-3632QM with 2.2GHz clock. The machine ran under control of a 64-bit Manjaro 0.8.8 GNU/Linux operating system with the GCC 4.8.2, glibc 2.18 and the Linux kernel 3.10.22-1-MANJARO.

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 4.2, update 2 [2]. OpenBLAS 0.2.8 [3] was linked for BLAS operations.

We used 8 threads, on the 4 cores, which means hyper-threading was used on all cores. According to the author's experiences, it reduces the computation time by a factor of c.a. 0.9 with respect to having a single thread per core. Please note that parallelization does not affect the number of iterations, but the execution time only.

The following test problems were considered – four of them were underdetermined (Academic, Hippopede, Puma6, 5R planar) and five – well-determined (Box3, Bratu30, Brent10, Broyden16, Transistor).

The first of the underdetermined ones is a set of two equations – a quadratic one and a linear one – in five variables [13]. It is called the Academic problem.

$$\begin{aligned}
&x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 1.0 = 0\,,\\
&x_1 + x_2 + x_3 + x_4 + x_5 = 0\,,\\
&x_1, x_2 \in [-1, 1],\ x_3 \in [-0.7, 0.7],\ x_4 \in [-0.8, 0.8],\ x_5 \in [-2, 2]\,.
\end{aligned} \tag{3}$$

Accuracy $\varepsilon = 0.05$

The second one is called the Hippopede problem [25, 32] – two equations in three variables.

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

Accuracy $\varepsilon = 10^{-7}$ was set.

The third problem, called Puma, arose in the inverse kinematics of a 3R robot and is one of typical benchmarks for nonlinear system solvers [6].

$$\begin{aligned}
&x_1^2 + x_2^2 - 1 = 0\,,\ x_3^2 + x_4^2 - 1 = 0\,,\\
&x_5^2 + x_6^2 - 1 = 0\,,\ x_7^2 + x_8^2 - 1 = 0\,,\\
&0.004731x_1x_3 - 0.3578x_2x_3 - 0.1238x_1 - 0.001637x_2 - 0.9338x_4 + x_7 = 0\,,\\
&0.2238x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0\,,\\
&x_6x_8 + 0.3578x_1 + 0.004731x_2 = 0\,,\\
&-0.7623x_1 + 0.2238x_2 + 0.3461 = 0\,,\\
&x_1, \ldots, x_8 \in [-1, 1]\,.
\end{aligned} \tag{5}$$

In the above form it is a well-determined (8 equations and 8 variables) problem with 16 solutions that are easily found by several solvers. To make it underdetermined the last equation was dropped – as in [25] – resulting in 7 equations with 8 variables. Accuracy $\varepsilon = 10^{-7}$ was set.

The fourth one is the inverse-kinematics problem of a planar redundant N-R manipulator, the effector of which should be placed in position $\left(1.0, 1.0, \frac{\pi}{2}\right)$. We presented the problem in Section 1, already, but we repeat it here for the sake of completeness:

$$\sum_{i=1}^{N} l_i \cdot \prod_{j=1}^{i} \cos\left(\sum_{k=1}^{j} x_k\right) - 1 = 0\,, \tag{6}$$
$$\sum_{i=1}^{N} l_i \cdot \prod_{j=1}^{i} \sin\left(\sum_{k=1}^{j} x_k\right) - 1 = 0\,,$$
$$\sum_{i=1}^{N} x_i - \frac{\pi}{2} = 0\,,$$
$$x_i \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \quad i = 1, \ldots, N\,.$$

We use this problem for $N = 5$, $l_i = 1$, $i = 1, \ldots, 5$; the accuracy is set to $\varepsilon = 2 \cdot 10^{-2}$.

The fifth problem is well-determined – it is called Box3 [6] and has three equations in three variables.

$$\exp(-0.1 \cdot x_1) - \exp(-0.1 \cdot x_2) - x_3 \cdot (\exp(-0.1) - \exp(-1.0)) = 0\,, \tag{7}$$
$$\exp(-0.2 \cdot x_1) - \exp(-0.2 \cdot x_2) - x_3 \cdot (\exp(-0.2) - \exp(-2.0)) = 0\,,$$
$$\exp(-0.3 \cdot x_1) - \exp(-0.3 \cdot x_2) - x_3 \cdot (\exp(-0.3) - \exp(-3.0)) = 0\,.$$
$$x_1, x_2 \in [-100.0, 100.0],\ x_3 \in [0.1, 100.0]\,.$$

Accuracy ε was set to 10^{-5}.

The sixth problem is well-determined, also and very sparse; it is called Bratu [6].

$$\frac{\exp(x_1)}{N+1} - 2x_1 + x_2 = 0\,, \tag{8}$$
$$x_{i-1} + \frac{\exp(x_i)}{N+1} - 2x_i + x_{i+1} = 0, \quad i = 2, \ldots, N-1\,,$$
$$x_{N-1} + \frac{\exp(x_N)}{N+1} - 2x_N = 0\,,$$
$$x_i \in [-10^8, 20], \quad i = 1, \ldots, N\,.$$

We consider this problem for size $N = 30$. Accuracy $\varepsilon = 10^{-6}$ was set.

The seventh problem is called the Brent problem – it is a well-determined algebraic problem, supposed to be "difficult" [4].

$$3x_1 \cdot (x_2 - 2x_1) + \frac{x_2^2}{4} = 0\,, \tag{9}$$
$$3x_i \cdot (x_{i+1} - 2x_i + x_{i-1}) + \frac{(x_{i+1} - x_{i-1})^2}{4} = 0, \quad i = 2, \ldots, N-1\,,$$
$$3x_N \cdot (20 - 2x_N + x_{N-1}) + \frac{(20 - x_{N-1})^2}{4} = 0\,,$$
$$x_i \in [-10^8, 10^8], \quad i = 1, \ldots, N\,.$$

Presented results have been obtained for $N = 10$; accuracy was set to 10^{-7}.

The eight one is the well-known Broyden-banded system [6, 25].

$$x_i \cdot (2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j \cdot (1 + x_j) = 0, \quad i = 1, \ldots, N, \tag{10}$$
$$J_i = \{j \mid j \neq i \text{ and } \max\{1, i-5\} \leq j \leq \min\{N, i+1\}\},$$
$$x_i \in [-100, 101], \quad i = 1, \ldots, N.$$

In this paper we consider the case of $N = 16$. The accuracy $\varepsilon = 10^{-6}$ was set. And the last one we call "Transistor" is taken from [34]. It is a system of 9 equations in 9 variables:

$$\begin{aligned}
&(1 - x_1x_2) \cdot x_3 \cdot \Big(\exp\Big(x_5 \cdot (g_{1k} - g_{3k} \cdot 10^{-3} \cdot x_7 - g_{5k} \cdot 10^{-3} \cdot x_8)\Big) - 1\Big) + \\
&-g_{5k} + g_{4k} \cdot x_2 = 0, \qquad k = 1, \ldots, 4, \\
&(1 - x_1x_2) \cdot x_4 \cdot \Big(\exp\Big(x_6 \cdot (g_{1k} - g_{2k} - g_{3k} \cdot 10^{-3} \cdot x_7 + g_{4k} \cdot 10^{-3} \cdot x_9)\Big) - 1\Big) + \\
&-g_{5k} \cdot x_1 + g_{4k} = 0, \qquad k = 1, \ldots, 4, \\
&x_1 \cdot x_3 - x_2 \cdot x_4 = 0. \\
&x_i \in [0, 10], \quad i = 1, \ldots, 9.
\end{aligned} \tag{11}$$

The matrix of g_{mk} parameters can be found, e.g., in [34] and [33]. Accuracy $\varepsilon = 10^{-8}$ was used in our experiments.

The following notation is used in the tables:

- fun.evals, grad.evals, Hesse evals – numbers of functions evaluations, its gradients and Hesse matrices evaluations (in the interval automatic differentiation arithmetic),
- bisecs – the number of boxes bisections,
- preconds – the number of preconditioning matrix computations (i.e., performed Gauss-Seidel steps),
- bis.Newt, del.Newt – numbers of boxes bisected/deleted by the Newton step,
- Sobol excl. – the number of boxes to be excluded generated by the initial exclusion phase,
- Sobol resul. – the number of boxes resulting from the exclusion phase, i.e., the size of the box-set L to be considered by the B&P method,
- bc3 – the number of calls of the consistency enforcing algorithm – Algorithm 3,
- bc3.rev. – the number of "first-level" calls (i.e., not counting the recursive ones) of "left_narrow" and "right_narrow" procedures,
- del.bc3 – the number of boxes deleted by consistency enforcing,
- q.solv – the number of interval quadratic equations the algorithm was trying to solve,
- q.del.delta – the number of boxes deleted, because the discriminant of the quadratic equation was negative,
- q.del.disj. – the number of boxes deleted, because the solutions of a quadratic equation were disjoint with the original box,
- q.bisecs – the number of boxes bisected by the quadratic equations solving procedure,

- pos.boxes, verif.boxes – number of elements in the computed lists of boxes containing possible and verified solutions,
- Leb.pos., Leb.verif. – total Lebesgue measures of both sets,
- time – computation time in seconds.

The ultimate table – Table 14, showing results for the currently most efficient algorithm version – has two additional rows, describing speedups with respect to two reference versions:

- sp(basic) – with respect to version "basic+BLAS" (see below for the description of both names),
- sp(PPAM) – with respect to version "PPAM2011".

We present results for the following algorithm versions:

- basic – for each box we compute the Jacobi matrix and use the interval Gauss-Seidel step with inverse-midpoint preconditioner; bisection over the variable with maximal diameter; no additional tools,
- basic+BLAS – as above, but the inverse-midpoint preconditioner is computed approximately and BLAS procedures are applied for matrix operations,
- PPAM2011 – the version presented in [27],
- PPAM2011+BC3(ε_{bc3}),
- PPAM2011+QH – the version presented in [27], with the Hansen's quadratic test, but no Sobol exclusion phase; see [29],
- PPAM2011+Sobol(k) – the version presented in [28], but with the new complement computing algorithm,
- PPAM2011+BC3(ε_{bc3})+QH, PPAM2011+Sobol(k)+BC3(ε_{bc3}), etc. – various combinations of the used tools.

Tables 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12 contain results for several simple versions of the algorithm, using many variants of the used tools. The two ultimate tables – Tables 13 and 14 contain the experiments for versions that –according to previous experiments – occurred to be most promising (see the analysis in the next section).

Remark 1 Please note that Table 1 contains two sets of results – the additional row ("BLAS–time") presents the computation times of the "basic+BLAS" algorithm version. This is done to save space. Other quantities, i.e., numbers of function evaluations, gradients, etc. are not presented for this version as they are very similar to results for the "basic" version. Very similar, but not identical – minor differences can be observed for problems Hippopede, 5R planar and Box3. Details are available form the author upon request.

Remark 2 Please note, results of the exclusion phase in its current version are not deterministic. The number in the filed "Sobol resul." may vary by a small factor and also computational time may be a few seconds higher or lower (also the number of

Table 1 Computational results for the "basic" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5099958	15430096	127006635	41619276	1862967	n/a	692155860	3037787520	n/a
grad.evals	5459166	17730164	142572087	46624515	2413323	n/a	1332804310	4741431712	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1364383	4414539	9918975	7670082	401852	n/a	63957959	145335937	n/a
preconds	2549979	7715048	18143805	13873092	620989	n/a	69215586	189861720	n/a
pos.boxes	963966	2243236	4349048	2213873	0	n/a	460	0	n/a
verif.boxes	78	49240	730212	2875	1	n/a	820	1	n/a
Leb.poss.	0.025456	2e-16	2e-46	0.000201	0.0	n/a	2e-83	0.0	n/a
Leb.verif.	6e-6	0.002829	3e-11	9e-7	2e-32	n/a	5e-66	2e-144	n/a
time	8	21	250	81	4	>25200	2604	21852	>25200
BLAS–time	8	20	188	67	4	>25200	1842	12641	>25200

Table 2 Computational results for the "PPAM2011" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5791240	1184688	15454537	52026620	2322997	n/a	47045208	7975494792	n/a
grad.evals	4970062	1361178	17188227	44904339	2191527	n/a	72183620	2139405200	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1242177	329911	1175559	7443520	365226	n/a	3107316	66082093	n/a
preconds	2252934	591826	2196460	11994885	501105	n/a	2944499	24741064	n/a
pos.boxes	918242	149952	469476	2034683	0	n/a	479	0	n/a
verif.boxes	80	21672	134904	5360	1	n/a	816	1	n/a
Leb.poss.	0.026040	1e-17	3e-47	0.000213	0.0	n/a	3e-83	0.0	n/a
Leb.verif.	1e-5	0.003696	3e-12	2e-6	2e-28	n/a	2e-78	3e-119	n/a
time	7	2	23	65	3	>25200	97	6112	>25200

Table 3 Computational results for the "PPAM2011+QH" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5831102	126897	12625413	44092631	1937796	n/a	20700598	3401716583	n/a
grad.evals	5085861	140166	13976050	41202560	1997370	n/a	22823633	967104330	n/a
Hesse evals	66307	554	3847	959024	524463	n/a	692853	179498	n/a
bisections	1254497	27761	954575	6632902	245462	n/a	732681	29441878	n/a
preconds	2271690	63293	1792964	11379669	403596	n/a	331628	6791151	n/a
q.solv.	326875	1074	7510	4352161	1012464	n/a	1728762	1025037	n/a
pos.boxes	930408	6305	367488	2029353	0	n/a	412	0	n/a
verif.boxes	106	9515	119754	5215	1	n/a	831	1	n/a
Leb.poss.	0.025587	1e-18	1e-47	0.000217	0.0	n/a	7e-81	0.0	n/a
Leb.verif.	6e-6	0.00171	4e-12	2e-6	3e-30	n/a	1e-77	7e-142	n/a
time	8	1	19	65	4	>25200	37	2664	>25200

Table 4 Computational results for the “PPAM2011+BC3$\left(\frac{3.0}{n}\right)$” algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5014338	87819	24125112	44268259	1552291	1231853	22240077	1250440729	n/a
grad.evals	5153082	77306	24554945	43045333	1724039	420146	31161132	188837774	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1287273	16479	1679770	7056642	268992	124	1140244	1517403	n/a
preconds	2411370	29500	2995327	12547020	410207	38	1297148	988510	n/a
bc3	659	15	1061	85419	1703	255	2167	2218194	n/a
bc3.rev.	11294	125	77162	1648904	115657	534730	1681913	554718137	n/a
pos.boxes	923876	2668	673988	2030583	0	0	508	3	n/a
verif.boxes	132	5900	191926	5274	1	2	815	0	n/a
Leb.poss.	0.026034	4e-19	7e-47	0.000212	0.0	0.0	6e-80	3e-118	n/a
Leb.verif.	9e-6	0.002	2e-11	2e-6	9e-14	4e-58	9e-28	0.0	n/a
time	8	1	33	63	2	3	43	644	>25200

Table 5 Computational results for the "PPAM2011+BC3($\frac{6.0}{n}$)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	4946610	890197	19485800	41961755	1497072	1227066	25531065	893365957	n/a
grad.evals	5100238	965698	19692918	43154412	1707192	418610	37540777	207266269	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1274991	233007	1341474	7134440	271551	130	1439622	4594351	n/a
preconds	2387038	402932	2387653	12578152	416994	48	1643007	3507489	n/a
bc3	31	3	457	9757	521	249	1643	800836	n/a
bc3.rev.	360	25	30334	181899	79817	531977	1605236	233294472	n/a
pos.boxes	912804	96608	513996	2033832	0	0	480	2	n/a
verif.boxes	136	23392	172046	5289	1	2	817	1	n/a
Leb.poss.	0.025508	6e-18	2e-47	0.000212	0.0	0.0	1e-81	4e-133	n/a
Leb.verif.	1e-5	0.00241	2e-11	2e-6	2e-10	4e-58	1e-24	3e-201	n/a
time	8	2	27	63	2	3	52	637	> 25200

Table 6 Computational results for the "PPAM2011+BC3($\frac{3.0}{n}$)+QH" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5006478	888373	24794224	41599644	1721475	1231853	10158847	2035680872	n/a
grad.evals	5162747	986070	25262074	40678742	2223871	420116	8589477	297420744	n/a
Hesse evals	13197	1032	1735	836324	695647	30	321778	2290286	n/a
bisections	1286435	240283	1728688	6505751	236380	124	59465	2625234	n/a
preconds	2409644	414184	3080977	11787318	466695	38	90966	1322624	n/a
bc3	655	15	1069	89634	1703	255	2131	3934205	n/a
bc3.rev.	11184	125	77008	1686330	115546	534730	1672994	882933577	n/a
q.solv.	65677	2058	3456	3916268	1333409	0	763891	13741063	n/a
pos.boxes	923020	107072	691982	2024307	0	0	402	2	n/a
verif.boxes	122	17436	198366	5186	1	2	819	0	n/a
Leb.poss.	0.026096	4e-18	7e-47	0.000216	0.0	0.0	6e-83	4e-111	n/a
Leb.verif.	1e-5	0.00212	2e-11	2e-6	5e-13	4e-58	2e-27	0.0	n/a
time	8	2	34	65	4	3	16	1117	> 25200

Table 7 Computational results for the "PPAM2011+Sobol(n)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5507856	1026155	11107650	55211483	2375774	n/a	38921242	8893494686	n/a
grad.evals	4755822	1176352	12218213	43839213	2286522	n/a	57116110	2330482768	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1188759	282037	832599	7260756	381053	n/a	2505112	72473310	n/a
preconds	2156701	512489	1576882	11119151	528280	n/a	2391450	31368027	n/a
Sobol excl.	5	3	8	5	3	n/a	9	16	9
Sobol resul.	76	14	422	30	9	n/a	642	94	233
pos.boxes	880637	121619	321830	1951942	0	n/a	498	0	n/a
verif.boxes	59	26126	108398	5418	1	n/a	810	1	n/a
Leb.poss.	0.028232	1e-17	9e-48	0.00029	0.0	n/a	1e-84	0.0	n/a
Leb.verif.	4e-6	0.002355	1e-11	2e-6	7e-23	n/a	9e-73	4e-167	n/a
time	7	2	17	63	3	>25200	78	6818	>25200

Table 8 Computational results for the "PPAM2011+Sobol($2n$)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5829582	1708847	21440684	55933794	2327860	n/a	40908422	8050851746	n/a
grad.evals	5051182	1960128	24002370	42407169	2181873	n/a	60849330	2055557120	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1262526	480955	1645463	7021393	363613	n/a	2686206	6399056	n/a1
preconds	2291168	853829	3051640	10424451	498274	n/a	2621136	20569945	n/a
Sobol excl.	10	6	16	10	6	n/a	19	32	18
Sobol resul.	131	34	11107	97	33	n/a	1624	360	373
pos.boxes	930717	224137	662931	1865094	0	n/a	487	0	n/a
verif.boxes	95	18771	179992	4566	1	n/a	816	1	n/a
Leb.poss.	0.026126	8e-18	3e-47	0.000343	0.0	n/a	7e-79	0.0	n/a
Leb.verif.	5e-6	0.002615	2e-11	2e-6	6e-24	n/a	4e-75	1e-166	n/a
time	7	3	32	61	3	>25200	83	5916	>25200

Table 9 Computational results for the "PPAM2011+Sobol($3n$)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5859352	944541	20192578	56779577	2380547	n/a	37640138	7249215876	n/a
ggrad.evals	5030610	1081598	22552558	43112220	2283192	n/a	56914490	1843881936	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1256893	259007	1533963	7137257	380500	n/a	2540565	57509981	n/a
preconds	2271660	471604	2870098	10615077	526054	n/a	2519414	18072457	n/a
Sobol excl.	15	9	24	15	9	n/a	29	48	27
Sobol resul.	548	34	27777	230	28	n/a	9034	2048	350
pos.boxes	921857	111946	616641	1896823	0	n/a	468	0	n/a
verif.boxes	231	23574	166595	4500	1	n/a	829	1	n/a
Leb.poss.	0.026344	1e-17	2e-47	0.000334	0.0	n/a	4e-82	0.0	n/a
Leb.verif.	1e-5	0.00197	1e-11	2e-6	9e-29	n/a	3e-77	5e-229	n/a
time	7	2	30	62	3	>25200	77	5437	>25200

Table 10 Computational results for the "PPAM2011+Sobol(n)+BC3($\frac{3.0}{n}$)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	4879721	501074	22277584	43017782	1551367	1464546	25034166	274963881	n/a
grad.evals	4989879	540619	22550108	40473339	1722887	514987	33329063	38118570	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1245509	130601	1545759	6624196	269049	128	1147495	430471	n/a
preconds	2332694	227634	2791376	11609841	410267	28	1307358	628669	n/a
Sobol excl.	5	3	8	5	3	30	10	15	9
Sobol resul.	53	11	422	30	9	103	2182	38	233
bc3	1067	20	836	80337	1699	305	3066	561067	n/a
bc3.rev.	18866	170	49414	1650129	115014	632125	2176362	128872597	n/a
pos.boxes	894076	56075	619842	1940462	0	0	519	0	n/a
verif.boxes	201	10853	171470	5446	1	2	808	1	n/a
Leb.poss.	0.026754	4e-18	3e-47	0.000289	0.0	0.0	2e-80	0.0	n/a
Leb.verif.	7e-5	0.00345	7e-11	2e-6	5e-9	7e-60	2e-28	4e-150	n/a
time	7	1	31	60	2	3	46	146	>25200

Table 11 Computational results for the "PPAM2011+Sobol($2n$)+BC3($\frac{3.0}{n}$)" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	4884148	907095	31466533	41958198	1360893	2061022	25529931	239257145	n/a
grad.evals	4983040	1007030	32139884	38437996	1535967	715625	33919961	33123407	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1243355	246527	2189400	6284336	241007	204	1150139	366714	n/a
preconds	2327115	427306	3898722	10896875	367054	65	1314861	523634	n/a
Sobol excl.	10	6	16	10	6	60	20	31	18
Sobol resul.	111	33	11107	97	33	160	17420	58	357
bc3	1247	26	2183	85043	1067	493	5918	473140	n/a
bc3.rev.	22335	235	70167	1731659	91043	899772	2236629	112283441	n/a
pos.boxes	891140	111710	857682	1854702	0	0	537	0	n/a
verif.boxes	496	14109	262186	4570	1	2	780	1	n/a
Leb.poss.	0.026915	6e-18	1e-47	0.000339	0.0	0.0	1e-82	0.0	n/a
Leb.verif.	8e-5	0.003021	5e-11	2e-6	4e-8	4e-57	4e-29	3e-89	n/a
time	7	2	43	57	2	3	47	127 >25200	

Table 12 Computational results for the "PPAM2011+Sobol($3n$)+BC3($\frac{3.0}{n}$)" algorithm version

Pproblem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	4838135	851775	29141525	42760753	1473813	2681752	24526223	221320116	n/a
grad.evals	4935341	939123	29445655	39133142	1642088	926269	34100264	30913609	n/a
Hesse evals	—	—	—	—	—	—	—	—	—
bisections	1231144	229403	1978884	6395159	256834	280	1169082	350526	n/a
preconds	2302227	398876	3518679	11101968	391932	82	1332289	503589	n/a
Sobol excl.	15	9	24	15	9	90	30	47	27
Sobol resul.	168	37	28164	230	28	195	87940	74	371
bc3	1314	31	4130	86325	1626	669	8136	443444	n/a
bc3.rev.	24029	275	101711	1793852	104555	1171128	2044853	103721508	n/a
pos.boxes	879956	103247	749844	1887092	0	0	524	0	n/a
verif.boxes	503	14684	264941	4510	1	2	801	1	n/a
Leb.poss.	0.027199	6e-18	2e-47	0.00033	0.0	0.0	6e-83	0.0	n/a
Leb.verif.	9e-5	0.00527	5e-11	2e-6	4e-9	4e-83	2e-30	3e-89	n/a
time	7	2	40	58	2	4	47	118	>25200

Table 13 Computational results for the "PPAM2011+Sobol(n^2)+BC3($\frac{6.0}{n}$)+QH" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5180255	515817	25141703	37093969	1646585	1773247	11743523	67477079	n/a
grad.evals	5338311	560306	25574671	37480368	2219816	595087	9963215	25586501	n/a
Hesse evals	11885	574	2435	889407	707094	30	347691	118160	n/a
bisections	1330939	135434	1749459	6012102	240656	191	62833	645031	n/a
preconds	2493927	235405	3127290	10711941	474731	78	97059	1019201	n/a
Sobol excl.	25	9	64	25	9	900	100	255	81
Sobol resul.	183	37	474	202	28	173	331	163	397
bc3	206	3	393	10831	478	438	1800	84768	n/a
bc3.rev.	3913	35	25215	233342	68755	748264	1843764	24905886	n/a
q.solv.	58783	1128	4802	4086396	1358323	0	827502	708619	n/a
pos.boxes	958980	58700	685579	1898019	0	0	404	0	n/a
verif.boxes	263	11234	209516	3957	1	2	832	1	n/a
Leb.poss.	0.025364	3e-18	3e-47	0.000308	0.0	0.0	2e-82	0.0	n/a
Leb.verif.	3e-5	0.00467	4e-11	2e-6	2e-8	4e-61	2e-30	1e-82	n/a
time	8	1	35	60	4	3	17	84	> 25200

Table 14 Computational results for the "PPAM2011+Sobol(n^2)+BC3($\frac{6.0}{n}$)+QH+NewBisHeur" algorithm version

Problem	Academic	Hippopede	Puma7	5R planar	Box3	Bratu30	Brent10	Broyden16	Transistor
fun. evals	5006186	566875	22517589	37819541	286618	1825254	10916862	56663975	151141488
grad.evals	5154649	620570	22857621	37967272	210336	614286	9256761	13695537	24899909
Hesse evals	11878	542	2455	901719	43952	30	346450	75695	1980029
bisections	1284673	150614	1562782	6085180	15648	163	63943	277906	405730
preconds	2404388	261482	2796246	10834714	30197	56	98483	437301	602844
Sobol excl.	25	9	64	25	9	900	100	255	81
Sobol resul.	239	37	384	295	28	151	298	167	397
bc3	289	3	381	15127	2308	449	1924	83067	560722
bc3.rev.	5300	35	25114	335983	83174	768749	1758928	24148338	69778092
q.solv.	58791	1067	4853	4142927	84589	0	824345	453448	5907859
pos.boxes	920148	66196	620398	1916117	0	0	423	0	0
verif.boxes	552	11337	188197	4399	1	2	810	1	1
Leb.poss.	0.026366	3e-18	3e-47	0.000295	0.0	0.0	1e-85	0.0	0.0
Leb.verif.	4e-5	0.00423	1e-10	2e-6	7e-12	1e-78	9e-28	5e-97	5e-50
time	8	1	31	61	1	3	17	49	89
sp(basic)	1.0	20.0	6.06	1.10	4.0	>8400.0	108.35	257.98	>283.15
sp(PPAM)	0.875	2.0	0.74	1.07	3.0	>8400.0	5.71	124.73	>283.15

iterations, etc., obviously). We do not emphasize it in the tables (nor present any statistical analysis of the phenomenon) as the uncertainty is minor.

7 Selected results obtained for Realpaver

For comparison, we present results for three test problems, obtained using another solver, *Realpaver* [7]. It is one of mature interval solvers that can be considered current state-of-the-art [17].

5R planar. For this underdetermined problem, Realpaver required 17 minutes (for `Bisection precision = 2.0`, much less accurate than the presented solver) and did not cover the whole solution set ("Property: non reliable process (some solutions may be lost)"). This result was far worse than ours.

Brent10. For this well-determined problem Realpaver has found all solutions (1065) in 55 seconds, but only when `-number 2000` was enforced. For default settings, it returned after 46 seconds with an incomplete list of boxes. Again, a worse result than for the presented solver.

Transistor. For this problem, Realpaver outperformed our solver. For the settings proposed by Realpaver authors (trisection and using weak 3B consistency; the benchmark is pre-defined in the configuration files of this solver), the solver requires 13 seconds, but does not verify the unique solution, but returns with a cluster of 3 boxes. For the default settings, Realpaver verifies the unique solution, but it takes 30 seconds – still a better result than our solver.

8 Analysis of results

Results, presented in Section 6 show that the performance impact of various tools may vary to the high extent. Using a single tool (initial exclusion phase, box consistency enforcing or Hansen's quadratic test) often improves the performance of the "PPAM2011" algorithm version. But applying two of the successful operations may result in far a smaller improvement or even in a slowdown. Apparently, some expensive tools might "redundant" when used together with other ones.

In particular, applying the Hansen's quadratic test improves the performance of the "PPAM2011" algorithm for problems Hippopede, Puma7 and Broyden16 (see Tables 2 and 3). But when we apply both BC3($3.0/n$) procedure and the quadratic test, results will be worse than for BC3 only (see Tables 4 and 6). Probably, the quadratic test (that requires Hesse matrix computation!) is applied for some boxes that could be reduced by the BC3 procedure, but it is difficult to verify this conjecture.

On the other hand, the Hansen's quadratic test improves the performance of all versions for the Brent10 problem.

In general, applying the BC3 procedure seems very worthwhile (improvements are dramatic for problems Brent10, Broyden16 and – particularly – Bratu30 that cannot be solved in a reasonable time without using the consistency operator), yet it is completely useless for the Puma7 problem (reasons for this behavior remain to be determined). The improvement for Box3 problem is minor, but irrefutable. Improvements for problems Academic and 5R-planar are minor (or none), but this seems to be related to the fact that these problems are underdetermined with the difference between the number of variables and equations of more than one (3 and 2, respectively; see below).

For two problems – Box3 and Transistor – it occurred to be crucial, to choose the proper coordinate for bisection, i.e., to use a heuristic related to MaxSmear, e.g., Algorithm 10.

The Transistor problem For this problem, only one algorithm version was able to provide the results. It was the most efficient version – results are presented in Table 14. Other experiments, not presented in this paper, show that for the Transistor problem, useful are only the following algorithm versions have the following properties, mutually:

– they use box-consistency,
– they use Algorithm 10.

The ultimate version gives the correct solution (the single box, guaranteed to contain the solution) in 89 seconds. Solvers, presented in [33] require 2359.5 seconds for the version tuned for this specific problem and 135099 for a more general version and in [16] – 444 seconds. These experiments have been performed on a Sun Ultra-2 running Solaris; according to [16], the clock frequency was 166MHz.

Our results are much better, but they are obtained on a far stronger machine, also. For using Realpaver, the correct solution was computed far quicker, on our machine – 30 seconds.

Apparently, the use of hull-consistency (that was used in [16] and also is incorporated in Realpaver) is pretty worthwhile. Unfortunately, hull-consistency enforcing is not easy to implement, especially in multithreaded environments. It requires complicated expression tree building and each thread should be able to traverse the tree (forward and backward) independently (intervals of values of respective quantities in the tree must be thread-specific). Still, the effort has to be done.

Underdetermined vs well-determined problems It is worth noting that tuning the algorithm for underdetermined problems occurred to be much harder than for well-determined ones. The Hippopede problem seems to be particularly “capricious” – results change rapidly for minor changes of algorithm features.

If the dimension of the solution set is higher than one, i.e., the difference between the number of variables and equations is higher than one, then tuning the algorithm does not have a significant impact on the performance. Comparing all tables shows that for such problems all algorithm versions perform similarly; it seems the ε we have to choose for such problems to stop in a reasonable time is so large, that specific features of various algorithm versions do not “have time” to affect the performance (or the time necessary to process all boxes containing solutions is too long). Experiments presented in the paper – problems Academic and 5R planar – but also in previous ones – problems Puma6 and Rheinboldt, see, e.g., [25–29] are consistent with this observation.

Also, underdetermined problems seem to require different policies for bisection than well-determined ones. In particular, heuristics based on smear computation, like MaxSmear, MaxSumMagnitude (e.g., [9]) perform particularly bad on them. It seems to be caused by the fact that *not all* of the components are going to be narrowed for underdetermined problems, but *only* the ones with the high smear, so the other components should be bisected, instead. If some components have been narrowed by the Newton operator, we should not bisect them (MaxSumMagnitudeUnnarrowed), but for boxes not narrowed yet, heuristics based on smear and magnitude cannot be applied at all. On the other hand, components with small smear and magnitude have minor impact on the system. So, it seems, bisecting the longest edge is the best solution and that is what we do in Algorithm 10.

The currently-best version Overall, the version that performs best, currently, occurred to be the one with the following features:

– the initial exclusion phase with n^2 Sobol points generated, sparsity-based expanding (see Section 4) and $N_{cutoff} = 128$,
– Algorithm 4 is used to decide whether to use BC3 or not, $\varepsilon_{bc3} = \frac{6.0}{n}$,
– the variable for bisection chosen by heuristic, described in Algorithm 10.

Results for this version are presented in Table 14.

9 Conclusions

Interval branch-and-prune solvers can use a great deal of tools to narrow and discard boxes. In this paper, the usefulness of some of them (proposed by the author and by other researchers) has been investigated. A proper heuristic to choose and parameterize the tools has been proposed.

In particular, we presented a novel initial exclusion phase and a new policy to choose the variable for bisection. This policy distinguishes underdetermined and well-determined problems, which seems another important novelty.

As test examples show, the proposed algorithm performs well and is successful for some hard problems (e.g., the Brent problem).

Comparison with the Realpaver solver imply that our solver can outperform it for underdetermined and non-typical (e.g., singular or ill-determined) problems, but performs much worse for the Transistor problem.

It is probably far from optimal and further research is going to be performed – in particular, applying machine learning techniques to self-tune the algorithm.

The source code of the presented version of the algorithm (and hopefully further versions) is going to be available at the author's page: https://www.researchgate.net/profile/Bartlomiej_Kubica?ev=hdr_xprf.

Future research Interesting results might be obtained, by applying AI methods to self-tune the heuristics. Up to now, the only paper investigating such an approach (but in a very limited version) is [15].

Also, hull consistency enforcing must be investigated (as we indicated earlier) – this procedure appears very useful in experiments performed by other researchers. So are the acceleration tools, proposed by Kolev [21].

Acknowledgements The author would like to thank Adam Woźniak and Sergey P. Shary for inspiring discussions.

Also, the author is grateful to the reviewers for helping to improve the paper by their valuable suggestions.

References

1. C++ extended scientific computing library. http://www.xsc.de (2013)
2. Intel Threading Building Blocks. http://www.threadingbuildingblocks.org (2013)
3. OpenBLAS library. http://xianyi.github.com/OpenBLAS/ (2013)
4. Difficult benchmark problems. http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node6.html (2014)
5. GNU linear programming kit.http://www.gnu.org/software/glpk/ (2014)
6. Non-polynomial nonlinear system benchmarks. https://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node2.html (2014)
7. Realpaver. nonlinear constraint solving & rigorous global optimization. http://pagesperso.lina.univ-nantes.fr/info/perso/permanents/granvil/realpaver/ (2014)
8. Sobol sequence generator. http://web.maths.unsw.edu.au/fkuo/sobol/ (2014)
9. Beelitz, T., Bischof, C.H., Lang, B.: A hybrid subdivision strategy for result-verifying nonlinear solvers. Tech. Rep. 04/8, Bergische Universitiät Wuppertal (2004)
10. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising hull and box consistency. In: International Conference on Logic Programming, pp. 230–244. The MIT Press (1999)
11. Benhamou, F., McAllester, D., Hentenryck, P.V.: CLP(intervals) revisited. In: Logic Programming, Proceedings of the 1994 International Symposium, pp. 124–138. The MIT Press (1994)
12. van Emden, M.H.: Computing functional and relational box consistency by structured propagation in atomic constraint systems. arXiv:preprint cs/0106008 (2001)
13. Goldsztejn, A., Jaulin, L.: Inner and outer approximations of existentially quantified equality constraints. Lect. Notes Comput. Sci. **4204**, 198–212 (2006)
14. Goualard, F.: On considering an interval constraint solving algorithm as a free-steering nonlinear Gauss-Seidel procedure. In: Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05, pp.1434–1438, ACM, New York, NY, USA (2005). URL http://doi.acm.org/10.1145/1066677.1067004
15. Goualard, F., Jermann, C.: A reinforcement learning approach to interval constraint propagation. Constraints **13**(1-2), 206–226 (2008)
16. Granvilliers, L., Benhamou, F.: Progress in the solving of a circuit design problem. J. Glob. Optim. **20**(2), 155–168 (2001)
17. Granvilliers, L., Benhamou, F.: Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. ACM Trans. Math. Softw. (TOMS) **32**(1), 138–156 (2006)
18. Hansen, E., Walster, W.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (2004)
19. Ishii, D., Goldsztejn, A., Jermann, C.: Interval-based projection method for under-constrained numerical systems. Constraints **17**(4), 432–460 (2012)

20. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
21. Kolev, L.V.: An improved interval linearization for solving nonlinear problems. Numeri. Algorithm. **37**(1-4), 213–224 (2004)
22. Kubica, B.J.: Intel TBB as a tool for parallelization of an interval solver of nonlinear equations systems. Tech. Rep. 09-02, ICCE WUT (2009)
23. Kubica, B.J.: Performance inversion of interval Newton narrowing operators. Prace Naukowe Politechniki Warszawskiej. Elektronika **169**, 111–119 (2009). KAEiOG 2009 Proceedings
24. Kubica, B.J.: Shared-memory parallelization of an interval equations systems solver – comparison of toos. Prace Naukowe Politechniki Warszawskiej. Elektronika **169**, 121–128 (2009). KAEiOG, 2009 Proceedings
25. Kubica, B.J.: Interval methods for solving underdetermined nonlinear equations systems. Reliable Computing **15**, 207–217 (2011). SCAN 2008 Proceedings
26. Kubica, B.J.: Exclusion regions in the interval solver of underdetemined nonlinear systems. Tech. Rep. 12-01, ICCE WUT (2012)
27. Kubica, B.J.: Tuning the multithreaded interval method for solving underdetermined systems of nonlinear equations. Lect. Notes Comput. Sci. **7204**, 467–476 (2012). PPAM 2011 Proceedings
28. Kubica, B.J.: Excluding regions using Sobol sequences in an interval branch-and-prune method for nonlinear systems. Reliab. Comput. **19**(4), 385–397 (2014). SCAN 2012 Proceedings
29. Kubica, B.J.: Using quadratic approximations in an interval method for solving underdetermined and well-determined nonlinear systems. Lect. Notes Comput. Sci. **8385**, 623–633 (2014). PPAM 2013 Proceedings
30. Kubica, B.J., Woźniak, A.: Using the second-order information in Pareto-set computations of a multi-criteria problem. Lect. Notes Comput. Sci. **7134**, 137–147 (2012). PARA 2010 Proceedings
31. Meyn, K.H.: Solution of underdetermined nonlinear equations by stationary iteration methods. Numer. Math. **42**, 161–172 (1983)
32. Neumaier, A.: The enclosure of solutions of parameter-dependent systems of equations. In: Reliability in Computing, pp. 269–286. Academic Press (1988)
33. Puget, J.F., Hentenryck, P.V.: A constraint satisfaction approach to a circuit design problem. J. Glob. Optim. **13**(1), 75–93 (1998)
34. Ratschek, H., Rokne, J.: Experiments using interval analysis for solving a circuit design problem. J. Glob. Optim. **3**(4), 501–518 (1993)
35. Rheinboldt, W.C.: Computation of critical boundaries on equilibrium manifolds. SIAM J. Numer. Anal. **19**, 653–669 (1982)
36. Shary, S.P.: An interval linear tolerance problem. Autom. Remote. Control. **65**, 1653–1666 (2004)
37. Shary, S.P.: Finite-dimensional Interval Analysis. XYZ. Electronic book (in Russian). (accessed 2014.05.15) (2013). URL http://www.nsc.ru/interval/Library/InteBooks/SharyBook.pdf