Interval methods for solving various kinds of quantified nonlinear problems

Bartłomiej Jacek Kubica

Institute of Control and Computation Engineering
Warsaw University of Technology
Poland

SCAN 2014
Würzburg

What shall we consider?

- What (in particular) kind of problems can we solve using interval methods?
- What is the algorithm to solve them?
- How to make this algorithm efficient?
- How to parallelize it successfully?

Why do we need interval analysis?

Why do we need interval analysis?

- Invented as a tool to bound/analyze numerical errors:
  - Ramon E. Moore, 1959 (1966).
  - Mieczysław Warmus, 1956.

Why do we need interval analysis?

- Invented as a tool to bound/analyze numerical errors:
  - Ramon E. Moore, 1959 (1966).
  - Mieczysław Warmus, 1956.
- Can be used for any other uncertainty, also:
  - discretization error, truncation error,
  - uncertain parameters,
  - decisions of other decision-makers,
  - ...

Why do we need interval analysis?

- Invented as a tool to bound/analyze numerical errors:
  - Ramon E. Moore, 1959 (1966).
  - Mieczysław Warmus, 1956.
- Can be used for any other uncertainty, also:
  - discretization error, truncation error,
  - uncertain parameters,
  - decisions of other decision-makers,
  - ...
- My view: the interval analysis is an approach to seek points, satisfying a certain logical condition.

Problem under solution

Find **all** $x \in \mathbb{R}^n$, satisfying the condition $P(x)$, i.e.,
find the set $\{x \in \mathbb{R}^n | P(x)\}$.

where *P (x)* is a predicate formula with a free variable *x*, i.e., free variables: $x_1, \ldots, x_n$.

It can contain bound variables, also (we shall call them "parameters").

Previous papers

- Computability of the problem:
  - V. Kreinovich, B. J. Kubica, *From computing sets of optima, Pareto sets and sets of Nash equilibria to general decision-related set computations*, Journal of Universal Computer Science, Vol. 16, pp. 2657 – 2685 (2010).
- Early presentation:
  - B. J. Kubica, *A class of problems that can be solved using interval algorithms*, SCAN 2010, Computing, Vol. 94 (2-4), pp. 271 – 280 (2012).

Explanation

- Several researchers investigated solving quantified constraints using the interval approach.
  - Papers of Ratschan, 2002; Benhamou & Goualard, 2000; Shary, 2002; Goldsztejn & Jaulin, 2006 (not exclusively!) are particularly notable.

$$\{x \in X \mid (\forall t \in [t_0, t_f]) \ (f(x,t) \leqslant 0)\}$$

Explanation

- Several researchers investigated solving quantified constraints using the interval approach.
  - Papers of Ratschan, 2002; Benhamou & Goualard, 2000; Shary, 2002; Goldsztejn & Jaulin, 2006 (not exclusively!) are particularly notable.

    $$\{x \in X \mid (\forall t \in [t_0, t_f])\ (f(x,t) \leqslant 0)\}$$
- Also, other problems, traditionally solved by interval methods, can be formulated that way, e.g.:
  - global optimization: $\{x \in X \mid (\forall t \in X)\ (f(x) \leqslant f(t))\}$
  - Pareto sets seeking,
  - ...

Example problems

$$\{x \in X \mid h(x) = 0\}$$

$$\{x \in X \mid f(x) \in [\underline{y}, \overline{y}]\}$$

$$\{x \in X \mid (\forall t \in X)\ (f(x) \leqslant f(t))\}$$

$$\{x \in X \mid (\forall t \in X)\ (\forall i = 1, \ldots, N\ \ f_i(x) \leqslant f_i(t)) \vee (\exists i\ f_i(x) < f_i(t))\}$$

$$\{x \in X \mid (\forall i = 1, \ldots, n)\ (\forall x'_i \in \boldsymbol{x_i} \subseteq \mathbb{R}^{k_i})\ (f_i(x_{\setminus i}, x'_i) \geqslant f_i(x))\}$$

where:

$$X \subseteq \mathbb{R}^n, \quad h : \mathbb{R}^n \to \mathbb{R}^m, \quad f, f_1, \ldots f_N : \mathbb{R}^n \to \mathbb{R}$$

Proposed algorithm

- We shall use the name generalized branch-and-bound method or branch-and-bound type method.
- Several algorithms, described in the literature, are its specific instances:
  - Branch-and-bound method.
  - Branch-and-prune method.
  - “Nested” b&b or b&p (for parameters).
  - SIVIA – Set Inversion Via Interval Analysis (Jaulin, 1993).
  - PPS – Partitioning Parameter Set (дроблене параметров; Калмыков, 1982).

Generic algorithm

Lpos = {}; *Lverif* = {}; *Lcheck* = {};
// Phase I
while (there are boxes to consider) **do**
pop (***x***);
process (***x***); // using interval tools
if (***x*** was verified to contain a solution/a point satisfying some necessary conditions) **then** push (*Lverif*, ***x***);
else if (***x*** is verified not to contain solutions) **then**
if (***x*** may be necessary in phase II) **then** push (*Lcheck*, ***x***);
else discard ***x*;**
end if
if (***x*** was discarded or stored) **then** pop (***x***)**;**
else if (diam (***x***) < ε**) then** push (*Lpos*, ***x***);
else
bisect (***x***, ***x1***, ***x2***); push (***x2***); ***x*** = ***x1***;
end if
end while
// Phase II – verification of *P(x)* for stored solution candidates
for each (***x*** in *Lverif* *Lpos*) **do**
if (***x*** does not contain a solution) **then** discard ***x***;
end for each

What does the algorithm result in?

- Two lists:
  - **`Lverif`** – the list of boxes verified (certified) to contain a solution,
  - **`Lpos`** – the list of boxes possibly containing a solution.
- What conditions have to be verified so that the solution was “verified”?
- There can be more than two lists, in general:
  - Some conditions are verified, some are not.
  - We classify points of the domain int more than two classes.

Algorithm's other details

- In what order do we process boxes? Does the order matter?
- How do we store boxes?
- What tools do we use to process a box?
- What information is needed to process a box in phase I?
- What information is needed to verify a box in phase II?
- In particular, what boxes do we store in *Lcheck* – if any?

Algorithm's other details

- All depends on the problem under solution.
- Does the presence of solution in one area have influence on its presence elsewhere?
  - For equations systems – not really.
  - For optimization problems – it does (the optimum occurs to be local, only, if we have found a better point elsewhere).
  - For seeking Pareto sets – also.
  - ...
- Obviously, rejection/reduction tests rely on the problem under solution, also.

How to store boxes?
The simplest case.

- The order of processing boxes does not matter.
  - Equations systems, CSPs, etc.
- Then, we can use any structure – a stack, queue, pool, etc.
- Parallelization is (relatively) easy.
- Some frameworks allow us to store the boxes implicitly, using the task queue (e.g., Intel TBB).
- Sufficient (not necessary!) condition: no bound occurrences of the variable x (i.e., no quantifiers range over its domain).

How to store boxes? More sophisticated cases.

- For, e.g., global optimization, the order, in which we process boxes, matters – and we know in what way.
- A priority queue is suitable.
- Parallelization is more difficult (but TBB has a class `priority_queue`).
- For some problems the order matters, but in a complicated way, difficult to predict.
  - E.g., Nash equilibria computing.
  - In my implementations, I used ordinary queues.
  - A potential for improvement, certainly.

Tools to process a box

- Order of function approximation:
  - 0th order tools – comparing function values.
  - 1st order tools – use of gradients.
  - 2nd order tools – use of Hesse matrices.
  - Higher order tools ?
- Operations:
  - Simply, comparing function values.
  - Several versions of the interval Newton operator (*componentwise*, GS) – on various levels.
  - Various constraint satisfaction methods (consistency enforcing, SIVIA, etc.).
  - ...

The information stored

- For equations systems, CSPs, etc. – no additional info is needed – boxes only (i.e., only local information).
- Global optimization – upper bound on the global minimum; a single number; no *Lcheck* list needed.
- Pareto sets seeking – we store the approximation of the Pareto frontier; a set of several points, changing frequently.
  - Tricky – especially for parallel implementations.
  - Simple 0^{th} order tools are inefficient.
  - Still, the list *Lcheck* is not needed (only a set of points in the criteria space).

The information stored

- A difficult problem – seeking Nash points or strong Nash points of a continuous game.
- To verify a solution, we need to compare players' utilities for boxes located in proper places of the search domain.
  - For “plain” Nash points – in proper subspaces; for strong ones – in the whole domain.
- An interval tree is an option (suggested in my earlier papers).
- Storing the boxes on the *Lcheck* list seems a better solution.

Three important example problems

- Seeking Nash points and strong Nash points of a continuous game.
- Seeking all local (including global) optima of a function.

My papers on games solutions seeking

- B. J. Kubica, A. Woźniak, *An interval method for seeking the Nash equilibria of non-cooperative games*, PPAM 2009, LNCS, Vol. 6068, pp. 446 – 455 (2010).
- B. J. Kubica, A. Woźniak, *Applying an interval method for a four agent economy analysis*, PPAM 2011, LNCS, Vol. 7204, pp. 477 – 483 (2012).
- B. J. Kubica, A. Woźniak, *Interval methods for computing various refinements of Nash equilibria*, SCAN 2012, unpublished.
- B. J. Kubica, A. Woźniak, *Interval methods for computing strong Nash equilibria of continuous games*, SING10 (2014), submitted to *Operations Research and Decisions*.

Solution concepts

John Forbes Nash Jr.
Nash equilibrium, 1950

Robert John (Yisrael) Aumann.
Strong Nash equilibrium, 1959

Picture from: http://commons.wikimedia.org/wiki/File:John_Forbes_Nash,_Jr._and_Robert_Aumann.jpg

Computing the solutions

- Nash points can use be defined by the following system of conditions:

$$\forall i = 1, \ldots, n \quad \forall x_i \in \boldsymbol{x_i} \subseteq \mathbb{R}^{k_i}$$

$$q_i\left(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_n^*\right) \geqslant q_i\left(x_1^*, \ldots, x_n^*\right)$$

- Hence, strong Nash points:

$$\forall I \subseteq \{1, \ldots, n\} \quad \forall x_I \in \boldsymbol{x_I} \subseteq \mathbb{R}^{k_i}$$

$q_i\left(x_{\setminus I}^*, x_I\right)$ does not dominate $q_i\left(x_1^*, \ldots, x_n^*\right)$

where x_I denotes a subvector, consisting of components of x with indexes from the set I.

Necessary conditions for Nash points

- Well determined ($N = \sum_i k_i$ equations and total N variables).
- The Jacobi matrix is composed of rows of Jacobi matrices of systems $\nabla q_i(x_1, \ldots, x_n) = 0$

$$
\begin{array}{cccc}
\dfrac{\partial q_1(x)}{\partial x_1} = 0, & \dfrac{\partial q_1(x)}{\partial x_2} = 0, & \cdots & \dfrac{\partial q_1(x)}{\partial x_n} = 0, \\
\dfrac{\partial q_2(x)}{\partial x_1} = 0, & \dfrac{\partial q_2(x)}{\partial x_2} = 0, & \cdots & \dfrac{\partial q_2(x)}{\partial x_n} = 0, \\
\vdots & \vdots & \cdots & \vdots \\
\dfrac{\partial q_n(x)}{\partial x_1} = 0, & \dfrac{\partial q_n(x)}{\partial x_2} = 0, & \cdots & \dfrac{\partial q_n(x)}{\partial x_n} = 0.
\end{array}
$$

Necessary conditions for Nash points

- Well determined ($N = \sum_i k_i$ equations and total N variables).
- The Jacobi matrix is composed of rows of Jacobi matrices of systems $\nabla q_i(x_{1,} \ldots, x_n) = 0$

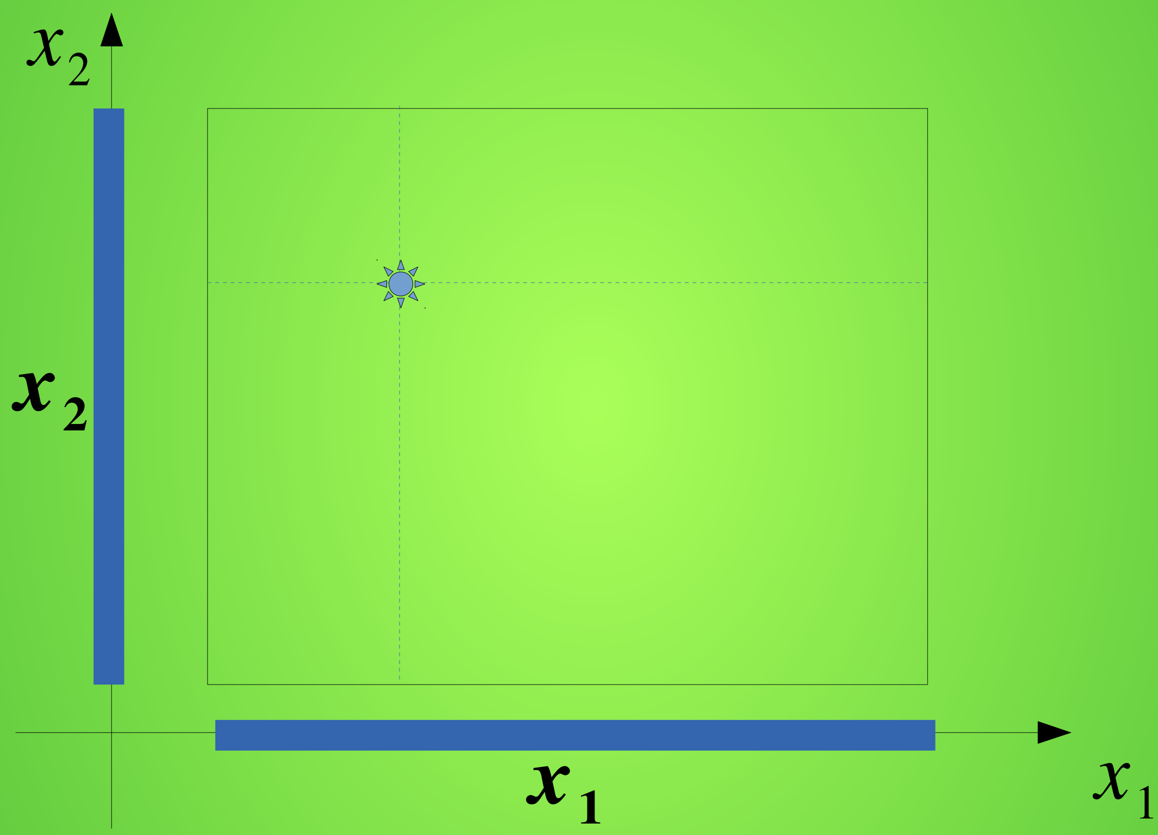
$$
\begin{array}{cccc}
\dfrac{\partial q_1(x)}{\partial x_1} = 0, & \xcancel{\dfrac{\partial q_1(x)}{\partial x_2}} = 0, & \cdots & \xcancel{\dfrac{\partial q_1(x)}{\partial x_n}} = 0, \\
\xcancel{\dfrac{\partial q_2(x)}{\partial x_1}} = 0, & \dfrac{\partial q_2(x)}{\partial x_2} = 0, & \cdots & \xcancel{\dfrac{\partial q_2(x)}{\partial x_n}} = 0, \\
\vdots & \vdots & \cdots & \vdots \\
\xcancel{\dfrac{\partial q_n(x)}{\partial x_1}} = 0, & \xcancel{\dfrac{\partial q_n(x)}{\partial x_2}} = 0, & \cdots & \dfrac{\partial q_n(x)}{\partial x_n} = 0.
\end{array}
$$

Necessary conditions for strong Nash points

- Necessary conditions:
  - All conditions for ordinary Nash points hold!
  - And there are additional ones.
- So, the system is overdetermined.
  - That is the reason (at least one of them) why SNEs exist so rarely.
  - It will not be possible to compute verified results using the interval Newton operator.
- What are these necessary conditions, specifically?
- We assume *i*-th player controls the variable $x_i \in X_i \subseteq \mathbb{R}$; extension to the general case is straightforward.

What do we check in phase II?

Consider the case with two players and $x_{1,}\, x_2 \in \mathbb{R}$

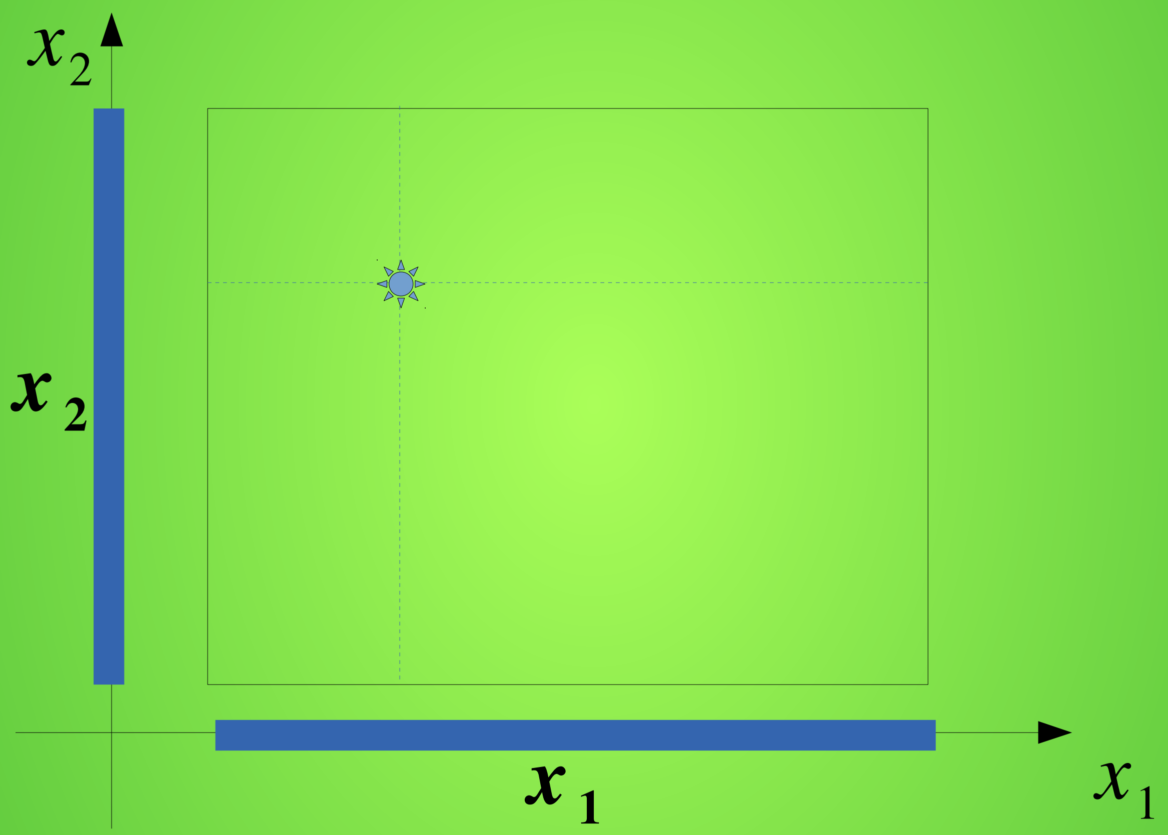


$$\forall\, x_1 \in \boldsymbol{x_1} \quad q_1(x_{1,} x_2^*) \geqslant q_1(x_1^*, x_2^*),$$

$$\forall\, x_2 \in \boldsymbol{x_2} \quad q_2(x_1^*, x_2) \geqslant q_2(x_1^*, x_2^*),$$

What do we check in phase II?

Consider the case with two players and $x_{1,} x_2 \in \mathbb{R}$



$$\forall\, x_1 \in \boldsymbol{x}_{1,} \quad x_2 \in \boldsymbol{x}_2 \qquad q_1(x_{1,} x_2) \geqslant q_1(x_1^*, x_2^*) \text{ or } x_1 \neq x_1^*,$$

$$q_2(x_{1,} x_2) \geqslant q_2(x_1^*, x_2^*) \text{ or } x_2 \neq x_2^*$$

Third important example: seeking local optima

$$\{x \in X \mid (\exists \delta > 0) \wedge (\forall t \in X \wedge d(x,t) < \delta)\ (f(x) \leqslant f(t))\}$$

- Find all local (and global) minima (or maxima).
- Should not be confused, e.g., with seeking an ε-optimal solution! $\{x \in X \mid (\forall t \in X)\ (f(x) \leqslant f(t) + \varepsilon)\}$
- The problem was rarely considered, up to now (and usually for very specific cases):
  - K. Villaverde, V. Kreinovich, *A linear-time algorithm that locates local extrema of a function of one variable from interval measurement results*, Interval Computations 4, pp. 176 – 194 (1993).
  - E. Lyager, *Finding local extremal points by using parallel interval methods*, Interval Computations, Vol. 3, pp. 63 – 80 (1994).
  - Ch. Eick, K. Villaverde, *Robust algorithms that locate local extrema of a function of one variable from interval measurement results: A remark*, Reliable Computing, Vol 2(3), pp. 213 – 218 (1996).

Third important example: seeking local optima

$$\{x \in X \mid (\exists \delta > 0) \wedge (\forall t \in X \wedge d(x,t) < \delta)\ (f(x) \leqslant f(t))\}$$

- Important potential applications:
  - Potential games (no pun intended) – local optima of the potential function are Nash equilibibira.
  - NMR spectroscopy – local maxima of the spectrum show, for which frequency the nucleus resonates.
  - Radio-astronomy – local maxima of the ray show where astronomical objects are located.
  - …
- How is the problem related to global optimization?

Comparison

Seeking global optima

- Efficient 0^{th} order tools (comparing function values).
- Global information on the minimum's upper bound.
- Boxes with smaller lower bound should be processed earlier.
- Phase II simple, but necessary.

Seeking local optima

- No 0^{th} order tools (function values are irrelevant).
- No global information.
- Order of boxes processing – irrelevant.
- No phase II.

- 1^{st} and higher order tools – very similar.

How to make the branch-and-bound-type method efficient?

How to make the branch-and-bound-type method efficient?

- There is a great deal of interval tools.
- All of them give guaranteed (verified) results.
- None of them are intelligent *per se*!

How to make the branch-and-bound-type method efficient?

- There is a great deal of interval tools.
- All of them give guaranteed (verified) results.
- None of them are intelligent *per se*!
- It is crucial to develop a heuristic to:
  - choose the interval tools adequate for a specific box,
  - arrange them,
  - parameterize them.

How to make the branch-and-bound-type method efficient?

- The author devoted several papers to design heuristics for two problems:
  - Nonlinear equations systems – especially seeking all solutions of underdetermined systems.
  - Seeking Pareto sets of a multicrietia problem.
- Many tools & versions; several papers.
- We present two topics:
  - Choosing the coordinate for bisection.
  - Initial exclusion phase – for nonlinear systems.

Underdetermined systems

- B. J. Kubica, *Interval methods for solving underdetermined nonlinear equations systems*, SCAN 2008, Reliable Computing, Vol. 15, pp. 207 – 217 (2011).
- B. J. Kubica, *Performance inversion of interval Newton narrowing operators*, KAEiOG 2009, Zeszyty Naukowe PW. Elektronika, Vol. 169, pp. 111 – 119 (2009).
- B. J. Kubica, *Shared-memory parallelization of an interval equations systems solver – comparison of tools*, KAEiOG 2009, *ibidem*, pp. 121 – 128.
- B. J. Kubica, *Intel TBB as a tool for parallelization of an interval solver of nonlinear equations systems*, ICCE internal report, 09-02, 2010.
- B. J. Kubica, *Tuning the multithreaded interval method for solving underdetermined systems of nonlinear equations*, PPAM 2011, LNCS, Vol. 7204, pp. 467 – 476 (2012).
- B. J. Kubica, *Excluding regions using Sobol sequences in an interval branch-and-prune method for nonlinear systems*, SCAN 2012, Reliable Computing, Vol. 19 (4), pp. 385 – 397 (2014).
- B. J. Kubica, *Using quadratic approximations in an interval method of solving underdetermined and well-determined nonlinear systems*, PPAM 2013, LNCS 8385, pp. 623 – 633 (2014).
- B. J. Kubica, *Presentation of a highly tuned multithreaded interval solver for underdetermined and well-determined nonlinear systems. Empirical evaluation of innovations*, Numerical Algorithms, submitted.

Pareto sets computing

- B. J. Kubica, A. Woźniak, *Interval methods for computing the Pareto-front of a multicriterial problem*, PPAM 2007, LNCS, Vol. 4967, pp. 1382 – 1391 (2008).
- B. J. Kubica, A. Woźniak, *A multi-threaded interval algorithm for the Pareto-front computation in a multi-core environment*, PARA 2008 Proceedings, LNCS, Vol. 6126 (not published, yet???).
- B. J. Kubica, A. Woźniak, *Optimization of the multi-threaded interval algorithm for the Pareto-set computation*, Journal of Telecommunications and Information Technology, Vol. 1, pp. 70 – 75 (2010).
- B. J. Kubica, A. Woźniak, *Using the second-order information in Pareto-set computations of a multi-criteria problem*, PARA 2010 Proceedings, LNCS, Vol. 7134, pp. 137 – 148 (2012).
- B. J. Kubica, A. Woźniak, *Computing Pareto-sets of multicriteria problems using interval methods*, presented at SCAN 2010, unpublished.
- B. J. Kubica, A. Woźniak, *Tuning the interval algorithm for seeking Pareto sets of multi-criteria problems*, PARA 2012, LNCS, Vol. 7782, pp. 504 – 517 (2013).

Underdetermined systems – tools used in my solver

- Interval Newton operators – we switch between the componentwise and Gauss-Seidel operators; a proper heuristic to choose.
  - BC3 for large (a heuristic to tell, which are large!) boxes.
- Using a quadratic approximation for boxes likely to contain singular points or otherwise hard for the Newton operator; a heuristic to decide.
- Initial exclusion phase, using 0^{th} order information, only.
- Two advanced policies to choose the bisection direction.

Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.

Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.
- An example of such heuristic is MaxSmear (Shary, 1992; Ratz, 1992; Ratz & Csendes, 1995).
  - Works very well for optimization problems.
  - Works reasonably well for well-determined equations systems.
  - Fails miserably for underdetermined systems.

Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.
- An example of such heuristic is MaxSmear (Shary, 1992; Ratz, 1992; Ratz & Csendes, 1995).
  - Works very well for optimization problems.
  - Works reasonably well for well-determined equations systems.
  - Fails miserably for underdetermined systems.
- In my opinion, the objective of bisection should be defined in a different way: give boxes that are **easy to process** by the used **interval tools**.

Bisection

- For equations solving, the main tool is some kind of the interval Newton operator.
- So, for a single equation in two variables, it might seem reasonable to choose the minimal smear.
- But the convergence...
- A proper policy should take into account several criteria.
- For several, advanced tools, such a policy cannot be too simple...

Bisection

- For example, the heuristic of Kubica, 2012:

find index *j_max* and diameter *w_max* of the longest component;
find index *j_min* and diameter *w_min* of the shortest component;
find index *j_max_nonred* and diameter *w_max_nonred* of the longest component **not reduced** by the latest use of the Newton;
if ((Newton operator reduced no component) or (*w_max* > 1.5 * *w_max_nonred*)) **then return** *j_max*;
else if (*w_max_nonred* > 8 * *w_min*) **then return** *j_max_nonred*;
find index *j* and diameter *w* of the component with the smallest maximal absolute value in all rows of the Jacobi matrix;
if (*w* > 0.1) **then return** *j*;
else return *j_max_noned*;

- My new paper proposes a new, yet different heuristic...

Bisection

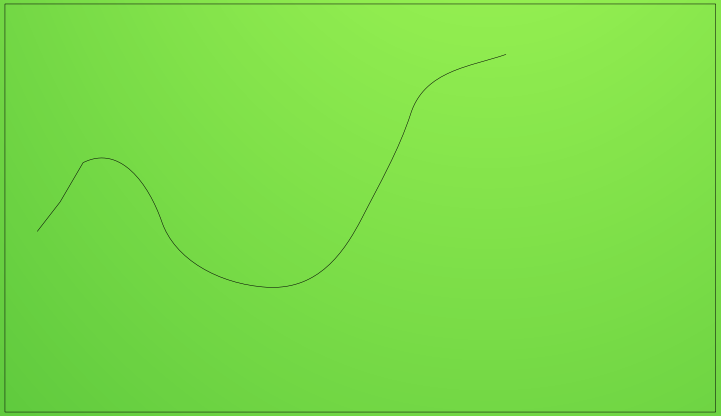
- For Pareto sets seeking, the proper heuristic is quite different:

find the index i of the criterion with maximal distance from the set in the criteria space;
find the index j and diameter w of the component with maximal smear with respect to criterion i;
find the index j_max and diameter w_max of the component with maximal diameter;
if ($w\_max < 8 * w$) **then return** j;
else return j_max;

- Reasons: different interval tools, used in the algorithm.

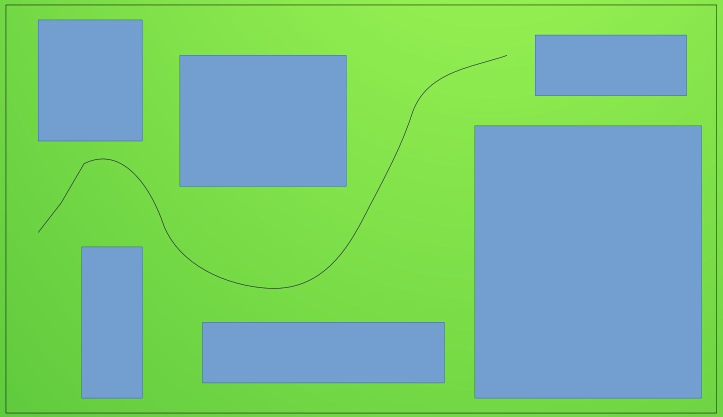
Underdetermined systems

- Initial exclusion phase – motivation:
  - Interval Newton operators are powerful, but relatively expensive.
  - Large boxes, encountered in the early stages of the b&p algorithm can rarely be reduced by the Newton operator.
  - We should apply these operators only for boxes close to the solution set.
  - Large regions of the domain can be discarded using function values, only.



Underdetermined systems

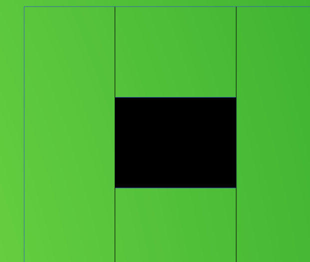
- Initial exclusion phase – motivation:
  - Interval Newton operators are powerful, but relatively expensive.
  - Large boxes, encountered in the early stages of the b&p algorithm can rarely be reduced by the Newton operator.
  - We should apply these operators only for boxes close to the solution set.
  - Large regions of the domain can be discarded using function values, only.



Underdetermined systems

- Initial exclusion phase – essence:
  - Before starting the actual branch-and-prune method, we generate a given number (n^2 ; n or $2n$ in earlier versions) of points, using the Sobol sequence.
  - Around the points we generate boxes, not containing solutions (procedure of Сергей П. Шарый for the linearized equation + ε-inflation; if $f(x) \in [-\varepsilon, \varepsilon]$, the point is ignored).
  - We exclude the boxes from the domain and start the b&p algorithm on their completion.
  - The procedure does not require using derivatives (global values are used for the Shary's procedure), so it is not computationally intensive.
  - Sobol sequences can be generated simply and efficiently (there are open libraries!).

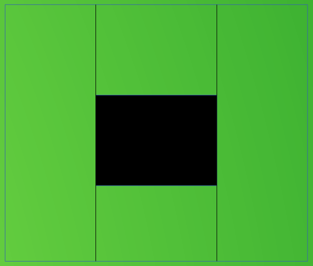
Underdetermined systems

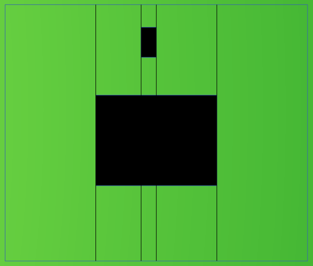


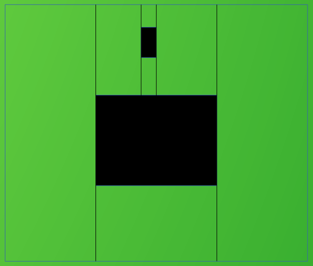
- Issue – proper implementation of the procedure computing the completion.
- There is the procedure of R.B. Kearfott for a single box; it generates at most 2 *n* boxes.
- It can be applied several times subsequently, but...

Underdetermined systems

- Issue – proper implementation of the procedure computing the completion.
- There is the procedure of R.B. Kearfott for a single box; it generates at most 2 *n* boxes.
- It can be applied several times subsequently, but:
  - It would not be parallel.
  - The result would depend on the order of boxes exclusion.
  - The generated box set could be very large.
  - Often, boxes have peculiar shapes (long and flat), their shapes are unrelated to function values.
  - Hence, actually, sometimes expanding the exclusion boxes decreases the performance.







Underdetermined systems

- Boxes might be sorted with respect to decreasing Lebesgue measure, but it solves the problem rarely.
- The satisfying solution:
  - We use task parallelism. Each task is to cut from a specific box a list of excluded boxes.
  - From this list we choose the box with the largest (wrt the Lebesgue measure) intersection with the box from which we do the exclusion.
  - Boxes, created in the exclusion process, become basis for new tasks (obviously, their lists of excluded boxes are shorter by one than for the parent task).
  - Far fewer boxes are created and the parallelism is natural.
  - All functions $f_i(\cdot)$ are used for exclusion.

Underdetermined systems

- For each function, after the ε-inflation, variables, not occurring in its formula, are set to their whole domain.
- We exclude the box for $f_i(\cdot)$, for which we obtained the largest Lebesgue measure.
- There is a threshold value not to exclude to many boxes (128 worked well, but it is a magical constant, obviously).

• Intel TBB allows an elegant implementation:

- We use the concept of **`tbb::parallel_do`**.
- Boxes, created in the exclusion process, become basis for new tasks – using **`tbb::parallel_do_feeder`**.
- Lists of boxes are represented as **`std::vector`** (**`tbb::concurrent_vector`** does not have the method **`pop_back`**).
- Counter of excluded boxes it represented as **`tbb::atomic`**.

Underdetermined systems – computational times

Problem	GS only	PPAM 2011	PPAM 2011 + BC3	Full algorithm
Broyden 16	21 851s (6h 4min 11s)	6112 s (101min 52s)	644 s (10min 44s)	70 s (1 min 10 s)
Bratu 30	broken > 7h	broken > 7h	3 s	4 s
Brent 10	2604 s (43min 24s)	97 s (1min 37s)	43 s	18 s
Hippopede	21 s	2 s	under 1 s	1 s
5R planar	81 s	65 s	63 s	59 s

Parallelization

- We already discussed that, but – to sum up.
- Parallelization is simplest, when:
  - the order of boxes processing is irrelevant and
  - no global information is needed.
- Such problems are, i.a.:
  - equations systems,
  - CPSs,
  - seeking all local optima.
- The box list can be stored implicitly – e.g., Intel TBB.
- Parallelization with OpenMP simple, also – particularly for OpenMP 3 (the **`task`** directive), but not only.

Shared memory

- For global optimization:
  - The order of boxes matters – a priority queue is needed.
  - Global information on the upper bound on the global minimum (a single floating-point number).
- Parallelization is more difficult, but the difference is minor:
  - Applying TBB not that natural (the class **`tbb::priority_queue`** might be useful!).
  - The global minimum upper bound should be protected by a mutex (or something similar).
- Pareto sets seeking:
  - We have to store the Pareto frontier – sounds scary...
  - Difficult choice of a box to process.

Shared memory

- Parallel Nash points seeking:
  - Several shared lists.
  - In my resent implementation the boxes are stored in a queue – can we do better?
  - Second phase is very important...
  - … and difficult – it requires a nested branch-and-bound procedure for each solution to (verify it).

- Local memory:
  - Box migration.
  - Termination detection.
  - Much more difficult if we need shared information.

Summary

- Interval methods can be applied for a wide class of problems, described by predicate formulae.
- Various kinds of these problems can be solved by some instances of the generalized branch-and-bound method.
- Details depend on the specific problem. They are difficult (or impossible) to determine automatically – a human is needed.
- For the efficiency, it is crucial to develop a proper heuristic to choose and parameterize the interval tools.
- The branch-and-bound-type methods parallelize well, but not trivially; some knowledge is needed to do it properly.