UPPSALA
UNIVERSITET

CoProD'16

9th International Workshop
for
Constraint Programming
and
Decision Making

Organized By

Martine Ceberio & Vladik Kreinovich
The University of Texas At El Paso

UPPSALA UNIVERSITET

THE UNIVERSITY OF TEXAS AT EL PASO

TIME:	PRESENTORS & ABSTRACTS
A.M. 8:00	**Welcome**
8:15	*"Thick separators"* **Luc Jaulin and Benoit Desrochers**
9:00	*"When We Know the Number of Local Maxima, Then We Can Compute All of Them"* **Olga Kosheleva, Martine Ceberio, and Vladik Kreinovich**
9:25	*"Using Interval Constraint Solving Techniques to Handle Large Nonlinear Simulations"* **Martine Ceberio & Leobardo Valera**
9:50	*"Balancing Waste Water Treatment Plant Load Using Branch and Bound"* **Ronald van Nooijen and Alla Kolechkina**
10:15	**Break-15min**
10:30	*"Collective Defense and Possible Relaxations in Weighted Abstract Argumentation Problems"* **Stefano Bistarelli**
11:15	*"How Neural Networks (NN) Can (Hopefully) Learn Faster by Taking Into Account Known Constraints"* **Chitta Baral, Martine Ceberio, and Vladik Kreinovich**
11:40	*"Construction of a Mosaic from an Underwater Video: an Interval Analysis Approach"* **M. Laranjeira, L. Jaulin, S. Tauvry, and C. Aubry**
12:05	**END OF THE WORKSHOP**

Canceled Presentation	*"Design of PI Controllers based on Constraint Programming using Frequency Envelope Condition"* **Harsh Purohit and P. S. V. Nataraj**

Organized By

Martine Ceberio & Vladik Kreinovich
The University of Texas At El Paso

Thick Separators

Luc Jaulin and Benoit Desrochers

Lab-STICC, ENSTA Bretagne, Brest, France,
luc.jaulin@gmail.com,benoit.desrochers@ensta-bretagne.org

Abstract. If an interval of $\mathbb{R}$ is an uncertain real number, a *thick set* is an uncertain subset of $\mathbb{R}^n$. More precisely, a thick set is an interval of the powerset of $\mathbb{R}^n$ equipped with the inclusion $\subset$ as an order relation. It can generally be defined by parameters or functions which are not known exactly, but are known to belong to some intervals. In this paper, we show how to use constraint propagation methods in order to compute efficiently an inner and an outer approximations of a thick set. The resulting inner/outer contraction are made using an operator which is called a *thick separator*. Then, we show how thick separators can be combined in order to compute with thick sets.

1 Thick sets

A thin set is a subset of $\mathbb{R}^n$. It is qualified as *thin* because its boundary is thin. In this section, we present the definition of thick sets.

Thick set. Denote by $(\mathcal{P}(\mathbb{R}^n), \subset)$, the powerset of $\mathbb{R}^n$ equipped with the inclusion $\subset$ as an order relation. A *thick set* $[\![\mathbb{X}]\!]$ of $\mathbb{R}^n$ is an interval of $(\mathcal{P}(\mathbb{R}^n), \subset)$. If $[\![\mathbb{X}]\!]$ is a thick set of $\mathbb{R}^n$, there exist two subsets of $\mathbb{R}^n$, called the *subset bound* and the *supset bound* such that

$$[\![\mathbb{X}]\!] = \qquad [\![\mathbb{X}^{\subset}, \mathbb{X}^{\supset}]\!] = \{\mathbb{X} \in \mathcal{P}(\mathbb{R}^n) | \mathbb{X}^{\subset} \subset \mathbb{X} \subset \mathbb{X}^{\supset}\}. \tag{1}$$

Another representation for the thickset $[\![\mathbb{X}]\!]$ is the partition $\left\{\mathbb{X}^{in}, \mathbb{X}^{?}, \mathbb{X}^{out}\right\}$, where

$$\begin{array}{rcl} \mathbb{X}^{in} & = & \mathbb{X}^{\subset} \\ \mathbb{X}^{?} & = & \mathbb{X}^{\supset} \backslash \mathbb{X}^{\subset} \\ \mathbb{X}^{out} & = & \overline{\mathbb{Z}^{\supset}}. \end{array} \tag{2}$$

The subset $\mathbb{X}^{?}$ is called the *penumbra* and plays an important role in the characterization of thick sets [3]. Thick sets can be used to represent uncertain sets (such as an uncertain map [4]) or a soft constraints [1].

2 Thick separators

To characterize a thin set using a paver, we may use a separator (which is a pair of two contractors [8]) inside a paver. Separators can be immediately generalized to

thick sets. Now, the penumbra as a non-zero volume for thick sets. For efficiency reasons, it is important to avoid any accumulation of the paving deep inside the penumbra. This is the role of thick separators to avoid as much as possible bisections inside the penumbra.

Thick separators. A *thick separator* $[\![\mathcal{S}]\!]$ for the thick set $[\![\mathbb{X}]\!]$ is an extension of the concept of separator to thick sets. More precisely, a thick separator is a 3-uple of contractors $\left\{\mathcal{S}^{in}, \mathcal{S}^{?}, \mathcal{S}^{out}\right\}$ such that, for all $[\mathbf{x}] \in \mathbb{IR}^n$

$$\begin{array}{c} \mathcal{S}^{in}([\mathbf{x}]) \cap \mathbb{X}^{in} = [\mathbf{x}] \cap \mathbb{X}^{in} \\ \mathcal{S}^{?}([\mathbf{x}]) \cap \mathbb{X}^{?} = [\mathbf{x}] \cap \mathbb{X}^{?} \\ \mathcal{S}^{out}([\mathbf{x}]) \cap \mathbb{X}^{out} = [\mathbf{x}] \cap \mathbb{X}^{out} \end{array} \tag{3}$$

In what follow, we define an algebra for thick separator in a similar manner than what as been done for contractors [2] or for separators [7].

3 Algebra

In this section, we show how we can define operations for thick sets (as a union, intersection, difference, etc.). The main motivation is to be able to compute with thick sets.

Intersection. Consider two thick sets $[\![\mathbb{X}]\!] = [\![\mathbb{X}^{\subset}, \mathbb{X}^{\supset}]\!]$ and $[\![\mathbb{Y}]\!] = [\![\mathbb{Y}^{\subset}, \mathbb{Y}^{\supset}]\!]$ with thick separators $[\![\mathcal{S}_{\mathbb{X}}]\!] = \left\{\mathcal{S}^{in}_{\mathbb{X}}, \mathcal{S}^{?}_{\mathbb{X}}, \mathcal{S}^{out}_{\mathbb{X}}\right\}$ and $[\![\mathcal{S}_{\mathbb{Y}}]\!] = \left\{\mathcal{S}^{in}_{\mathbb{Y}}, \mathcal{S}^{?}_{\mathbb{Y}}, \mathcal{S}^{out}_{\mathbb{Y}}\right\}$. A thick separator for the thick set

$$[\![\mathbb{Z}]\!] = [\![\mathbb{Z}^{\subset}, \mathbb{Z}^{\supset}]\!] = [\![\mathbb{X}]\!] \cap [\![\mathbb{Y}]\!] = [\![\mathbb{X}^{\subset} \cap \mathbb{Y}^{\subset}, \mathbb{X}^{\supset} \cap \mathbb{Y}^{\supset}]\!] \tag{4}$$

is

$$\begin{array}{rl} [\![\mathcal{S}_{\mathbb{Z}}]\!] = & \left\{\mathcal{S}^{in}_{\mathbb{Z}}, \mathcal{S}^{?}_{\mathbb{Z}}, \mathcal{S}^{out}_{\mathbb{Z}}\right\} \\ = & \left\{\mathcal{S}^{in}_{\mathbb{X}} \cap \mathcal{S}^{in}_{\mathbb{Y}}, \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{in}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{in}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right), \mathcal{S}^{out}_{\mathbb{X}} \sqcup \mathcal{S}^{out}_{\mathbb{Y}}\right\}. \end{array} \tag{5}$$

Proof. We have

$$\begin{array}{rrcl} \mathbb{Z}^{in} = & \mathbb{Z}^{\subset} & = & \mathbb{X}^{\subset} \cap \mathbb{Y}^{\subset} \\ & & = & \mathbb{X}^{in} \cap \mathbb{Y}^{in} \\ \mathbb{Z}^{?} = & \mathbb{Z}^{\supset} \backslash \mathbb{Z}^{\subset} & = & \mathbb{X}^{\supset} \cap \mathbb{Y}^{\supset} \backslash (\mathbb{X}^{\subset} \cap \mathbb{Y}^{\subset}) \\ & & = & \mathbb{X}^{\supset} \cap \mathbb{Y}^{\supset} \cap \overline{\mathbb{X}^{\subset} \cap \mathbb{Y}^{\subset}} \\ & & = & \mathbb{X}^{\supset} \cap \mathbb{Y}^{\supset} \cap \left(\overline{\mathbb{X}^{\subset}} \cup \overline{\mathbb{Y}^{\subset}}\right) \\ & & = & \left(\mathbb{X}^{in} \cup \mathbb{X}^{?}\right) \cap \left(\mathbb{Y}^{in} \cup \mathbb{Y}^{?}\right) \cap \left(\left(\mathbb{X}^{out} \cup \mathbb{X}^{?}\right) \cup \left(\mathbb{Y}^{out} \cup \mathbb{Y}^{?}\right)\right) \\ & & = & \left(\mathbb{X}^{?} \cap \mathbb{Y}^{in}\right) \cup \left(\mathbb{X}^{?} \cap \mathbb{Y}^{?}\right) \cup \left(\mathbb{X}^{in} \cap \mathbb{Y}^{?}\right) \\ \mathbb{Z}^{out} = & \overline{\mathbb{Z}^{\supset}} & = & \overline{\mathbb{X}^{\supset} \cap \mathbb{Y}^{\supset}} \\ & & = & \overline{\mathbb{X}^{\supset}} \cup \overline{\mathbb{Y}^{\supset}} \\ & & = & \mathbb{X}^{out} \cup \mathbb{Y}^{out}. \end{array}$$

From the separator algebra, we get that a contractor for $\mathbb{Z}^{in}$ is $\mathcal{S}^{in}_{\mathbb{Z}} = \mathcal{S}^{in}_{\mathbb{X}} \cap \mathcal{S}^{in}_{\mathbb{Y}}$, a contractor for $\mathbb{Z}^{out}$ is $\mathcal{S}^{out}_{\mathbb{Z}} = \mathcal{S}^{out}_{\mathbb{X}} \sqcup \mathcal{S}^{out}_{\mathbb{Y}}$ and a contractor for $\mathbb{Z}^{?}$ is

$$\mathcal{S}^{?}_{\mathbb{Z}} = \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{in}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{in}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right). \tag{6}$$

4 Using Karnaugh map

This expression could have been obtained using Figure 1.

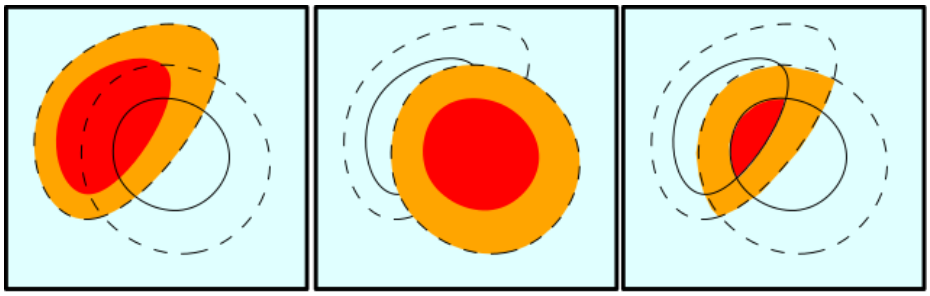


Fig. 1. Intersection of two thick sets. Red means *inside*, Blue means *outside* and Orange means *uncertain*

Karnaugh map, as illustrated by Figure 2, can also can be used to get the expression for thick separators a more clear manner. For instance, if

$$[\![\mathbb{Z}]\!] = [\![\mathbb{X}]\!] \cup [\![\mathbb{Y}]\!], \tag{7}$$

we read from the Karnaugh map

$$\begin{array}{rcl} \mathbb{Z}^{in} & = & \mathbb{X}^{in} \cup \mathbb{Y}^{in} \\ \mathbb{Z}^{?} & = & \left(\mathbb{X}^{?} \cap \mathbb{Y}^{out}\right) \cup \left(\mathbb{X}^{?} \cap \mathbb{Y}^{?}\right) \cup \left(\mathbb{X}^{out} \cap \mathbb{Y}^{?}\right) \\ \mathbb{Z}^{out} & = & \mathbb{X}^{out} \cap \mathbb{Y}^{out}. \end{array} \tag{8}$$

Therefore a thick separator for the thick set $[\![\mathbb{Z}]\!]$ is

$$\begin{array}{rl} [\![\mathcal{S}_{\mathbb{Z}}]\!] = & \left\{\mathcal{S}^{in}_{\mathbb{Z}}, \mathcal{S}^{?}_{\mathbb{Z}}, \mathcal{S}^{out}_{\mathbb{Z}}\right\} \\ = & \left\{\mathcal{S}^{in}_{\mathbb{X}} \sqcup \mathcal{S}^{in}_{\mathbb{Y}}, \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{out}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{out}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right), \mathcal{S}^{out}_{\mathbb{X}} \cap \mathcal{S}^{out}_{\mathbb{Y}}\right\} \end{array} \tag{9}$$

Now, if

$$[\![\mathbb{Z}]\!] = [\![\mathbb{X}]\!] \backslash [\![\mathbb{Y}]\!] \cup [\![\mathbb{Y}]\!] \backslash [\![\mathbb{X}]\!], \tag{10}$$

we read

$$\begin{array}{rcl} \mathbb{Z}^{in} & = & \left(\mathbb{X}^{in} \cap \mathbb{Y}^{out}\right) \cup \left(\mathbb{X}^{out} \cap \mathbb{Y}^{in}\right) \\ \mathbb{Z}^{?} & = & \mathbb{X}^{?} \cup \mathbb{Y}^{?} \\ \mathbb{Z}^{out} & = & \left(\mathbb{X}^{in} \cap \mathbb{Y}^{in}\right) \cup \left(\mathbb{X}^{out} \cap \mathbb{Y}^{out}\right). \end{array} \tag{11}$$

Therefore a thick separator for the thick set $[\![\mathbb{Z}]\!]$ is

$$\begin{array}{rl} [\![\mathcal{S}_{\mathbb{Z}}]\!] = & \left\{\mathcal{S}^{in}_{\mathbb{Z}}, \mathcal{S}^{?}_{\mathbb{Z}}, \mathcal{S}^{out}_{\mathbb{Z}}\right\} \\ = & \left\{\mathcal{S}^{in}_{\mathbb{X}} \sqcup \mathcal{S}^{in}_{\mathbb{Y}}, \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{out}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{?}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right) \sqcup \left(\mathcal{S}^{out}_{\mathbb{X}} \cap \mathcal{S}^{?}_{\mathbb{Y}}\right), \mathcal{S}^{out}_{\mathbb{X}} \cap \mathcal{S}^{out}_{\mathbb{Y}}\right\}. \end{array} \tag{12}$$

Note that when we build such an expression from a Karnaugh map, fake boundaries may appear. They could be avoided using the method proposed in [5].

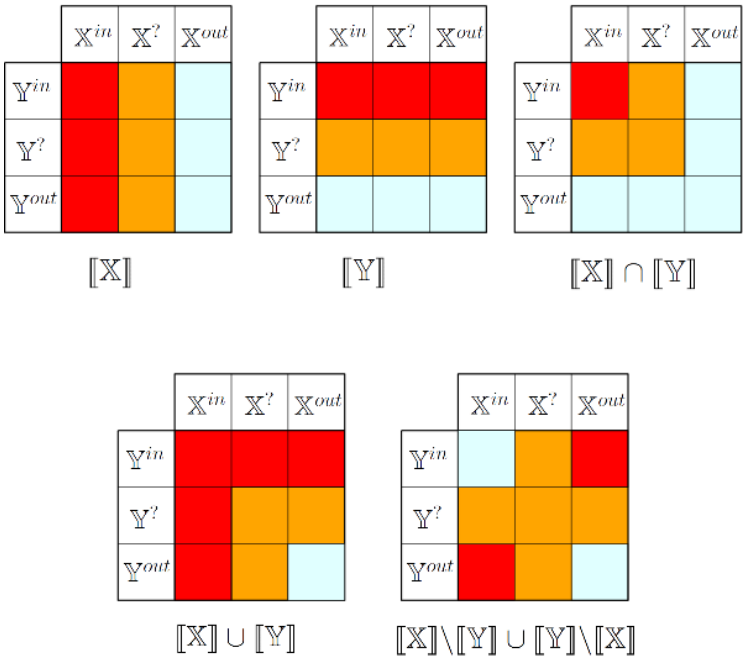


Fig. 2. Karnaugh map

Example. Take one box $[\mathbf{x}]$ as in Figure 3. We get

$$[\![\mathcal{S}_{\mathbb{X}}]\!]([\mathbf{x}]) = \left\{\mathcal{S}_{\mathbb{X}}^{in}, \mathcal{S}_{\mathbb{X}}^{?}, \mathcal{S}_{\mathbb{X}}^{out}\right\}([\mathbf{x}]) = \{[\mathbf{a}], [\mathbf{x}], \emptyset\} \tag{13}$$

where $[\mathbf{a}]$ the white box. Moreover,

$$[\![\mathcal{S}_{\mathbb{Y}}]\!]([\mathbf{x}]) = \left\{\mathcal{S}_{\mathbb{Y}}^{in}, \mathcal{S}_{\mathbb{Y}}^{?}, \mathcal{S}_{\mathbb{Y}}^{out}\right\}([\mathbf{x}]) = \{\emptyset, [\mathbf{x}], \emptyset\}\,. \tag{14}$$

Thus

$$\begin{aligned}
[\![\mathcal{S}_{\mathbb{Z}}]\!] &= \left\{\mathcal{S}_{\mathbb{Z}}^{in}, \mathcal{S}_{\mathbb{Z}}^{?}, \mathcal{S}_{\mathbb{Z}}^{out}\right\}([\mathbf{x}]) \\
&= \{\ \mathcal{S}_{\mathbb{X}}^{in} \cap \mathcal{S}_{\mathbb{Y}}^{in}([\mathbf{x}]), \\
&= \left(\mathcal{S}_{\mathbb{X}}^{?} \cap \mathcal{S}_{\mathbb{Y}}^{in}\right) \sqcup \left(\mathcal{S}_{\mathbb{X}}^{?} \cap \mathcal{S}_{\mathbb{Y}}^{?}\right) \sqcup \left(\mathcal{S}_{\mathbb{X}}^{in} \cap \mathcal{S}_{\mathbb{Y}}^{?}\right)([\mathbf{x}]), \\
&= \mathcal{S}_{\mathbb{X}}^{out} \sqcup \mathcal{S}_{\mathbb{Y}}^{out}([\mathbf{x}]) \qquad\qquad \} \\
&= \{\ [\mathbf{a}] \cap \emptyset, ([\mathbf{x}] \cap \emptyset) \sqcup ([\mathbf{x}] \cap [\mathbf{x}]) \sqcup ([\mathbf{a}] \cap [\mathbf{x}])\,, \emptyset \sqcup \emptyset\ \} \\
&= \{\emptyset, [\mathbf{x}], \emptyset\}
\end{aligned}$$

We conclude that $[\mathbf{x}] \subset \mathbb{Z}^{in}$.

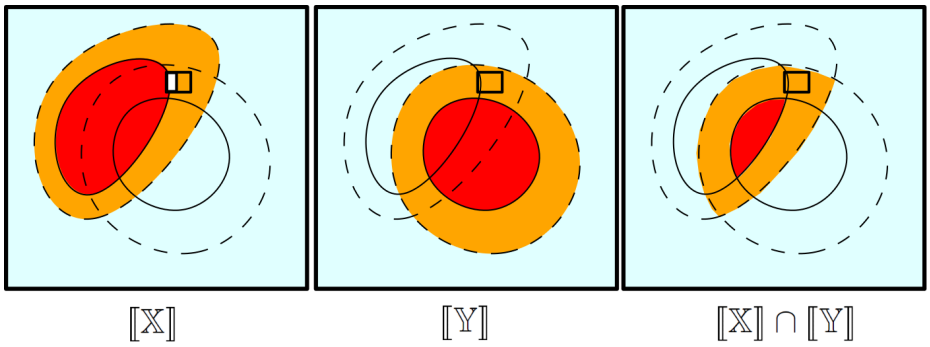


Fig. 3. Illustration of the intersection of two separators

5 Test case

Interval linear system [10] [11] are linear systems of equations the coefficients of which are uncertain and belong to some intervals. Consider for instance the following interval linear system [9]:

$$\begin{cases} [2,4] \cdot x_1 + [-2,0] \cdot x_2 \in [-1,1] \\ [-1,1] \cdot x_1 + [2,4] \cdot x_2 \in [0,2] \end{cases} \tag{15}$$

For each constraint, a thick separator can be build and then combined using Equation 5. A thick set inversion algorithm provides the paving Figure 4. The solution set $[\![\mathbb{X}]\!] = [\![\mathbb{X}^{\subset}, \mathbb{X}^{\supset}]\!]$ has for supset bound the *tolerable solution set* $\mathbb{X}^{\supset}$(red+orange) and for subset bound $\mathbb{X}^{\subset}$ the *united solution set* (red) [6]. Note that inside the penumbra, no accumulation can be observed.

References

1. Q. Brefort, L. Jaulin, M. Ceberio, and V. Kreinovich. If we take into account that constraints are soft,then processing constraints becomes algorithmically solvable. In *Proceedings of the IEEE Series of Symposia on Computational Intelligence SSCI'2014*. Orlando, Florida, December 9-12, 2014.
2. G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173:1079–1100, 2009.
3. B. Desrochers and L. Jaulin. Computing a guaranteed approximation the zone explored by a robot. *IEEE Transaction on Automatic Control*, 2016.
4. B. Desrochers, S. Lacroix, and L. Jaulin. Set-membership approach to the kidnapped robot problem. In *IROS 2015*, 2015.
5. G. Schvarcz Franco and L. Jaulin. How to avoid fake boundaries in contractor programming. In *SWIM'16*, 2016.
6. A. Goldsztejn and G. Chabert. On the approximation of linear ae-solution sets. In *12th International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Duisburg, Germany, (SCAN 2006)*, 2006.

7. L. Jaulin and B. Desrochers. Introduction to the algebra of separators with application to path planning. *Engineering Applications of Artificial Intelligence*, 33:141–147, 2014.
8. L. Jaulin and B. Desrochers. Robust localisation using separators. In *COPROD 2014*, 2014.
9. V. Kreinovich and S. Shary. Interval methods for data fitting under uncertainty: A probabilistic treatment. *Reliable Computing*, 2016.
10. S. Shary. On optimal solution of interval linear equations. *SIAM Journal on Numerical Analysis*, 32(2):610–630, 1995.
11. S. Shary. A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable Computing*, 8:321–418, 2002.

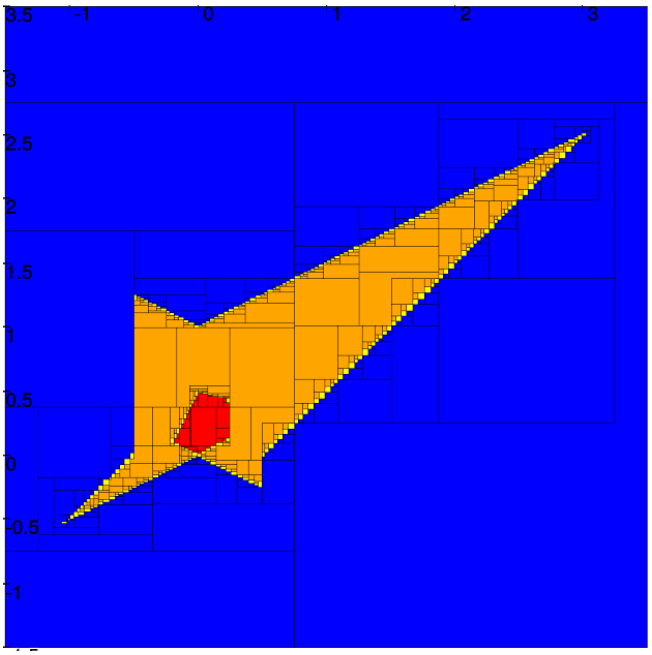


Fig. 4. Thick set corresponding to the test-case. Red boxes are inside the thick solution set and the blue boxes are outside. The penumbra corresponds to the orange boxes.

When We Know the Number of Local Maxima, Then We Can Compute All of Them

Olga Kosheleva, Martine Ceberio, and Vladik Kreinovich

University of Texas at El Paso, El Paso, TX 79968, USA
olgak@utep.edu, mceberio@utep.edu, vladik@utep.edu

Abstract. In many practical situations, we need to compute local maxima. In general, it is not algorithmically possible, given a computable function, to compute the locations of all its local maxima. We show, however, that if we know the *number* of local maxima, then such an algorithm is already possible. Interestingly, for global maxima, the situation is different: even if we only know the number of locations where the *global* maximum is attained, then, in general, it is not algorithmically possible to find all these locations. A similar impossibility result holds for local maxima if instead of knowing their exact number, we only know two possible numbers.

1 Locating Local Maxima: an Important Practical Problem

Need for computing local maxima. In many practical situations, we are interested in locating all local optima; see, e.g., [3]. For example:

- in spectral analysis, chemical species are identified by local maxima of the spectrum;
- in radioastronomy, radiosources and their components are identified as local maxima of the brightness distribution; see, e.g., [4];
- elementary particles are identified by locating local maxima of the dependence of scattering intensity on the energy.

In general, no algorithm is possible for computing all local maxima. In general, no algorithm is possible for computing all local maxima of a computable function $f(x)$; this follows, e.g., from our negative result formulated below.

Natural question and what we do in this paper. Since we cannot *always* compute all local maxima, a natural question is: *when* can we compute them? In this paper, we prove that such a computation is algorithmically possible in situations when we know the number of local maxima – and not possible if we only know two possible candidates for this number.

2 What Is Computable: Reminder

Need for a reminder. To formulate our results, we need to recall the main definitions of what is computable: what is a computable number, what is a computable set, and what is a computable function; see, e.g., [1, 2, 5] for more details.

What is a computable number: intuitive idea. In a computer, we can only represent rational numbers – namely, binary-rational ones. Thus, it is reasonable to say that a real number is computable if it can be algorithmically approximated, with any given accuracy, by rational numbers.

What is a computable number: a precise definition. A real number x is called *computable* if there is an algorithm that, given a natural number n, produces a rational number r_n which is 2^{-n}-close to x.

What is a computable set: intuitive idea. In a computer, we can only store finitely many objects – i.e., a finite set, with computable distances. It is therefore reasonable to define a computable set as a set that can be algorithmically approximated, with any given accuracy, by finite sets – approximated in the sense that every element of our set is 2^{-n}-close to one of the elements from the approximating finite set.

Since a computer has a linear memory, it is convenient to place all the elements of these finite sets – which approximate our set with higher and higher accuracy – into a single infinite sequence x_1, x_2, ... Elements from this sequence approximate any element from the given set. Thus, this sequence must be *everywhere* dense in this set.

In practice, we do not know the exact values of the elements, we only have approximations to elements of the set. Based on these approximations, we can never know whether the resulting set is closed or not – i.e., whether a set of real numbers is the interval $[-1, 1]$ or the same interval minus 0 point. To ignore such un-detectable differences, it is reasonable to assume that our set is *complete*, i.e., that it includes the limit of each converging sequence.

Thus, we arrive at the following definition.

What is a computable set: definition. By a *computable set*, we mean a complete metric space with an everywhere dense sequence $\{x_i\}$ for which:

- there is an algorithm that, given i and j, computes the distance $d(x_i, x_j)$ (with any given accuracy), and
- there exists an algorithm that, given a natural number n, returns a natural number $N(n)$ for which every point $x_1, x_2, \ldots$ is 2^{-n}-close to one of the points $x_1, \ldots, x_{N(n)}$.

By a *computable element* x of a computable set, we mean an algorithm that, given a natural number n, returns an integer $i(n)$ for which $d(x, x_{i(n)}) \le 2^{-n}$.

Comment. From the topological viewpoint, a complete metric space which can be approximated by finite sets is a *compact space*. Thus, computable sets are also known as *computable compact sets*.

What is a computable function: intuitive idea. A computable function f should be able, given a computable real number (or, more generally, a computable element of a computable set), to compute the value $f(x)$ with any given accuracy. Computable elements x are given by their approximations. Thus, to compute $f(x)$ with a given accuracy 2^{-n}, we need to:

- first algorithmically determine how accurately we need to compute x to achieve the desired accuracy 2^{-n} in $f(x)$, and then
- use the corresponding approximation to x to actually compute the desired approximation to $f(x)$.

So, we arrive at the following definition.

What is a computable function: definition. We say that a function $f(x)$ from a computable set to real numbers is computable if:

- first, we have an algorithm that, given n, returns m for which $d(x, x') \le 2^{-m}$ implies that $|f(x) - f(x')| \le 2^{-n}$, and
- second, we have an algorithm that, given i, computes $f(x_i)$.

Comment. The existence of m for every n is nothing else but uniform continuity; so, in effect, we want $f(x)$ to be effectively uniformly continuous.

Now, we are ready to formulate our main results.

3 Main Results

Proposition 1. *There exists an algorithm that:*

- *given an integer m and a computable function $f(x)$ with exactly m local maxima,*
- *always computes the locations of all these maxima.*

Comment 1. It is worth mentioning that for *global* maxima, such an algorithm is not possible even for $m = 2$: no algorithm can, given a computable function at which the global maximum is attained at exactly two points, computes these two locations; see, e.g., [2].

Comment 2. Knowing the exact number of local maxima is important: as the following result shows, if we have an *incomplete* information about this number, we can no longer compute all the local maxima.

Proposition 2. *Let $m < m'$ be two natural numbers. Then, no algorithm is possible that:*

- *given a computable function $f(x)$ with either m or m' local maxima,*
- *always returns the locations of all its local maxima.*

4 Proof of Proposition 1

Auxiliary results needed to describe our algorithm. Our algorithm is based on the several known results. The first is that we can algorithmically compute the maximum of a computable function on a computable set; see, e.g., [1, 2, 5].

We will also use an easy-to-prove fact that for every computable element x_0, the function $f(x) \stackrel{\text{def}}{=} d(x, x_0)$ is computable; see, e.g., [1, 5].

Another result that we will use is that for every computable function $f(x)$ on a computable set, and for every four rational numbers $\underline{r}_1 < \overline{r}_1 < \underline{r}_2 < \overline{r}_2$, we can algorithmically find the values $b_1 \in (\underline{r}_1, \overline{r}_1)$ and $b_2 \in (\underline{b}_2, \overline{b}_2)$ for which the set $\{x : b_1 \le f(x) \le b_2\}$ is also a computable set; see, e.g., [1].

We will also use the fact that each positive rational number p/q is simply a pair of natural numbers. Thus, a tuple consisting of natural and positive rational numbers can be viewed simply as a tuple consisting of natural numbers.

We can algorithmically sort the tuples consisting of positive natural numbers: e.g., first we consider all (finitely many) tuples whose sum is 1, then all the tuples whose sum is 2, etc.

Now, we are ready to describe our algorithm.

Our algorithm: a description. We want to locate all the maxima with a given accuracy 2^{-n}. To do that, by using one of above-mentioned sorting, we try, one by one, all possible tuples consisting of two natural numbers i and k and four positive rational numbers for which $\underline{r}_1 < \overline{r}_1 < \underline{r}_2 < \overline{r}_2 \le 2^{-n}$. For each such tuple, we compute $f \stackrel{\text{def}}{=} f(x_i)$ and $s \stackrel{\text{def}}{=} \max\{f(x) : b_1 \le d(x, x_i) \le b_2\}$ (for appropriate $b_i \in (\underline{r}_i, \overline{r}_i)$) with accuracy 2^{-k}. If for the resulting approximations $\widetilde{f}$ and $\widetilde{s}$, we get $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$, then we can conclude that $f > s$.

We stop when we get m different tuples such that:

- each of these m tuples satisfies the inequality $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$ (thus $f > s$), and
- for every two tuples, the distance $d(x_i, x_j)$ between the corresponding elements x_i and x_i' is larger than the sum of the corresponding values $\overline{r}_2$ and $\overline{r}_2'$.

The corresponding m elements x_i are then returned as the desired 2^{-n}-approximations to the locations of the local maxima.

What we need to prove. To prove the proposition, we need to prove:

- that this algorithm always stops, and that
- that this algorithm is correct, i.e., that the results of this algorithm are indeed 2^{-n}-approximations to the desired locations of local maxima.

Let us first prove that the algorithm always stops. Let $M_1, \ldots, M_m$ be the desired local maxima, and let d_0 be the smallest of all the distances between them.

By definition, a local maximum means that $f(M_j) \ge f(x)$ for all x from some neighborhood of M_j. We can always select this neighborhood of size $\le d_0/3$. This

way, we can be sure that there are no other local maxima in this neighborhood – and thus, no values $x \neq M_j$ with $f(M_j) = f(x)$, since otherwise these values x will also be local maxima.

Therefore, $f(M_j) > f(x)$ for all all the values from this neighborhood.

Let δ be a rational number which is smaller than all m radii of these neighborhoods. Then $f(M_j) > f(x)$ for all x for which $\delta/2 \le d(x, M_j) \le \delta$. The set $\{x : \delta/2 \le d(x, M_j) \le \delta\}$ is a compact, so for a continuous function $f(x)$, the maximum is attained at some element from this set. Since for all the points from this set, we have $f(x) < f(M_j)$, we therefore conclude that $f(M_j) > \max\{f(x) : \delta/2 \le d(x, M_j) \le \delta\}$.

Due to continuity, for elements x_i which are sufficiently close to M_j, we also have $f(x_i) > \max\{f(x) : \delta/2 \le d(x, M_j) \le \delta\}$. Here, if $d(x_i, M_j) \le \varepsilon$, then $\delta/2 + \varepsilon \le d(x, x_i) \le \delta - \varepsilon$ imply $\delta/2 \le d(x, M_j) \le \delta$. Thus:

$$\max\{f(x) : \delta/2 \le d(x, M_j) \le \delta\} \ge \max\{f(x) : \delta/2 + \varepsilon < d(x, x_i) \le \delta - \varepsilon\}$$

and

$$f(x_i) > \max\{f(x) : \delta/2 + \varepsilon < d(x, x_i) \le \delta - \varepsilon\}.$$

So, when we take $\underline{r}_1 = \delta/2 + \varepsilon$ and $\overline{r}_2 = \delta - \varepsilon$, we will get

$$f(x_i) > \max\{f(x) : b_1 < d(x, x_i) \le b_2\}.$$

Whenever the above strict inequality is true, we will detect it if we compute both sides of this inequality with sufficient accuracy. Thus, eventually, we will indeed find the tuples for which $\widetilde{f} > \widetilde{s} + 2 \cdot 2^{-k}$ and for which each x_i is desirably close to the corresponding local maximum M_j. Hence, our algorithm will indeed always stop.

Let us now prove that the algorithm is correct. To complete our proof, we need to show that when the algorithm stops, the resulting elements x_i are indeed close to the corresponding local maxima. Indeed, when the algorithm stops, for each of the selected m tuples, we get $f(x_i) > \max\{f(x) : b_1 \le d(x, x_i) \le b_2\}$.

On the compact set $\{x : d(x, x_i) \le b_2\}$, the maximum of the continuous function $f(x)$ is attained at some element from this set. Due to the above inequality, this maximum cannot be attained at distances between b_1 and b_2. Thus, this maximum is attained when $d(x, x_i) \le b_1 < b_2$. So, this maximum is a local maximum of the function $f(x)$, and a local maximum which is (due to $b_2 < \overline{r}_2 \le 2^{-n}$) 2^{-n}-close to the corresponding element x_i.

On each "zone" $\{x : d(x, x_i) \le b_2\}$ we thus have a local maximum of the given function $f(x)$. Since $d(x_i, x_j) > \overline{r}_2 + \overline{r}'_2 > b_2 + b'_2$, these zones do not intersect. Thus:

- all m corresponding local maxima are different,
- there are no local maxima outside these zones, and
- within each zone, we have exactly one local maximum which is 2^{-n}-close to x_i.

The correctness is proven, and so is the proposition.

5 Proof of Proposition 2

Our proof-by-contradiction is based on the fact that no algorithm is possible that, given a non-negative real number a, checks whether $a = 0$. This result can be easily proven based on the halting problem result. Indeed, for each Turing machine, we can define a computable real number a for which:

- $r_n = 2^{-n}$ if this Turing machine did not halt by moment n and
- $r_n = 2^{-t}$ if it halted at moment $t \le n$.

As a result:

- If the Turing machine does not halt, then the resulting number is equal to 0.
- Otherwise, if the Turing machine halts at some time t, then we have $a = 2^{-t} > 0$.

The impossibility to check whether a Turing machine halts implies that we cannot check whether $a = 0$.

For each m and m', let us define the following function $f(x)$ on the interval

$$[0, 2m + 2(m' - m) \cdot a] :$$

For $x \in [0, 2m]$, we have:

- $f(x) = 1 - |x - 1|$ for $0 \le x \le 2$,
- $f(x) = 1 - |x - 3|$ for $2 \le x \le 4$,
- $\ldots$,
- $f(x) = 1 - |x - (2m - 1)|$ when $2m - 2 \le x \le 2m$.

For $x \in [2m, 2m + 2(m' - m) \cdot a]$, we have:

- $f(x) = a - |x - (2m + a)|$ when $2m \le x \le 2m + 2a$,
- $f(x) = a - |x - (2m + 3a)|$ when $2m + 2a \le x \le 2m + 4a$,
- $\ldots$,
- $f(x) = a - |x - (2m + (2(m' - m) - 1) \cdot a)|$ when

$$2m + (2(m' - m) - 2) \cdot a \le x \le 2m + 2(m - m) \cdot a.$$

Here:

- When $a = 0$, this function has m local maxima, at points 1, 3, $\ldots$, $2m - 1$.
- When $a > 0$, this function has $m + (m' - m) = m'$ local maxima:
  - m local maxima at points 1, 3, $\ldots$, $2m - 1$, and
  - $m' - m$ local maxima at points $2m + a$, $2m + 3a$, $\ldots$, $2m + (2(m' - m) - 1) \cdot a$.

If we could always return all the local maxima, then by checking whether there is a local maximum close to $2m$, we would be able to check whether $a > 0$ or $a = 0$, and we have already shown that this is not possible. This proves Proposition 2.

Acknowledgments

This work was supported in part by NSF grants HRD-0734825, HRD-1242122, and DUE-0926721, and by an award from Prudential Foundation.

References

1. E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967.
2. V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.
3. K. Villaverde and V. Kreinovich, "A linear-time algorithm that locates local extrema of a function of one variable from interval measurement results," *Interval Computations*, 1993, No. 4, pp. 176–194.
4. G. L. Verschuur and K. I. Kellermann, *Galactic and Extra-Galactic Radio Astronomy*, Springer Verlag, Berlin, Heidelberg, New York, 1974.
5. K. Weihrauch, *Computable Analysis*, Springer Verlag, Berlin, 2000.

Using Interval Methods to handle Large Numerical Simulations

Martine Ceberio and Leobardo Valera

University of Texas at El Paso, El Paso, TX 79968, USA
mceberio@utep.edu, valera@utep.edu

Abstract. Many natural phenomena can be modeled as ordinary or partial differential equations. A way to find solutions of such equations is to discretize them and solve the corresponding (possibly) nonlinear large systems of equations; see [1]. Major issues with solving such nonlinear systems include the fact that their dimension can be very large and that uncertainty, often present, is tricky to handle. Model-Order Reduction (MOR) has been proposed as a way to overcome the issues associated with large dimensions, the most used approach for doing so being Proper Orthogonal Decomposition (POD); see [2,3]. The key idea of POD is to reduce a large number of interdependent variables (snapshots) of the system to a much smaller number of uncorrelated variables while retaining as much as possible of the variation in the original variables. On the other hand, interval constraint-solving techniques (ICST) [4] allow to handle uncertainty and ensure reliable results of systems of (possibly) nonlinear equations.
In this presentation, we show how intervals and constraint solving techniques (ICST) can be used to compute all the snapshots at once (IPOD). We take advantage of using interval techniques to also show how IPOD can address not only dimension but also uncertainty. As a result, this new process gives us two advantages over the traditional POD method: 1. handling uncertainty in some parameters or inputs; 2. reducing the snapshots computational cost. We go over numerical examples of the use of IPOD and we then take a glimpse at the implications of this work: How does using interval constraint solving techniques and handling uncertainty affect the whole simulation process? How is the reduced-order model then solved?

Keywords: dynamic systems, interval computations, constraint solving, proper-orthogonal decomposition (POD), reduced-order modeling (ROM).
Acknowledgment: Most of this work was supported by Stanfords Army High-Performance Computing Research Center (AHPCRC) funded by the Army Research Lab (ARL), and by the National Science Foundation award 0953339.

References:

[1] J. Li and Y. Chen, *Computational Partial Differential Equations Using MATLAB*. CRC Press, Las Vegas (NV), 2008.

[2] W. H. Schilders and H. A. Vorst, *Model Order Reduction: Theory, Research Aspects and Applications*. Springer Science & Business Media, 2008.

[3] K. Carlberg, C. Bou-Mosleh, and C. Farhat, *Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations*, Int. J. Numer. Meth. Engng., vol. 86, pp. 155–181, 2011.

[4] L. Granvilliers, and F. Benhamou, *RealPaver: An Interval Solver using Constraint Satisfaction Techniques*. ACM Trans. on Mathematical Software 32(1), 138156, 2006.

Balancing Waste Water Treatment Plant Load Using Branch and Bound

Ronald van Nooijen and Alla Kolechkina

Delft University of Technology
Stevinweg 1, 2628 CN Delft, Netherlands
{r.r.p.vannooyen,a.g.kolechkina}@tudelft.nl
http://wrm.tudelft.nl

Abstract. The problem of smoothing dry weather inflow variations for a Waste Water Treatment Plant (WWTP) that receives sewage from multiple mixed sewer systems is presented, together with a first rough solution algorithm. A simplification followed by a naive translation into a zero-one linear programming problem results in 1152 inequalities for 480 binary variables.

Keywords: Water Management, Branch and Bound

1 Introduction

In 2015 representatives of two water boards and one municipality decided to look into a long standing problem. The Garmerwolde Waste Water Treatment Plant (WWTP) receives water from several pressurized pipelines. These in turn receive sewage from local sewer systems through pumping stations. The WWTP plant was designed to process sewage as it arrives in the plant, there are no buffers. The supply varies roughly sinusoidally over a 24 hour period and the pumps are under local control (no coordination). This leads to extremely uneven supply to the WWTP, which in turn leads to high costs for chemical additives and air injection. Moreover, at times three large pumping stations may be using the same pressurized line, which wastes energy as well. Ideally, the flow to the WWTP should be approximately constant, the number of pump starts and stops should be limited, and the sewage in local sewer systems should not be stationary for long periods to avoid silting up of the pipes. The pumping stations themselves cannot deliver all flows between 0 and their maximum capacity, there may be "holes" in the flow range. In fact some of them may be limited to on/off operation, so this has some of the properties of a mixed integer problem. More information on the design and operation of Dutch combined sewer systems can be found in [2].

2 Abstract problem formulation

We have m reservoirs and an n time step inflow forecast for each reservoir. Time step length is Δt, t_k is the start of time step k. For each reservoir there is a time

dependent regulated outflow. We introduce the following functions (lower case: scalars or vectors, upper case: closed finite intervals):

$v_i(t)$ volume stored in reservoir i at time t, non-negative real number;
$\tau_i(t)$ time since last state change of pumping station i;
$\tau_{e,i}(t)$ time since reservoir i was last considered empty;
$V_i(k)$ limits on storage use for reservoir i at t_k, a non-negative, non-empty real closed finite interval;
$v_{e,i}$ level for which reservoir i is considered empty;
$q_{in,i}(t)$ actual inflow into reservoir i at time t, non-negative real number;
$q_{fc,i}(k)$ average inflow into reservoir i forecast for time step k, non-negative real number;
c_i number of different flow ranges available for pumping station i, $c_i \in \mathbb{N}$, $c_i > 0$;
$C_i(k)$ set of flow ranges accessible to pumping station i in time step k, integer, $\{0\} \subseteq C_i(k) \subseteq \{0, 1, \ldots, c_i\}$;
$j_{out,i}(t)$ actual flow range in use for pumping station i at time t, this is a piecewise constant function;
$j_i(k)$ selected flow range for pumping station i for time step k, non-negative real number;
$Q_{i,j}(k)$ with $j = 0, 1, 2, \ldots, c_i(k) - 1$, non-negative, non-empty disjoint real closed finite intervals, for all i and k we have $Q_{i,0}(k) = [0, 0]$ which is used when the pump is off;
$q_{out,i}(t)$ actual discharge from pumping station i at time t;
$q_i(k)$ discharge setting for pumping station i for time step k, non-negative real number;
$\tau_{min,i}$ minimum time between pump range switching moments for pump at reservoir i;
$\tau_{e,max,i}$ maximum time between times that $v_i(t) \leq v_{e,i}(k)$ for reservoir i;
$Q_{tgt}(k)$ range of allowed total flows to the WWTP in time step k.

3 Problem to be solved

The simplest version of the problem is the following. Given the initial state $v_i\left(t_0^-\right)$, $\tau_{e,i}\left(t_0^-\right)$, $\tau_i\left(t_0^-\right)$, $j_{out,i}\left(t_0^-\right)$ for $i = 1, 2, \ldots, m$; fixed parameters $\tau_{min,i}$, $v_{e,i}$ for $i = 1, 2, \ldots, m$ and the constraints up to the time horizon, $q_{in,i}(k)$, $C_i(k)$, $Q_{i,j}(k)$, $Q_{tgt}(k)$ for $i = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, n-1$. The constraint $V_i(k)$ is available for $i = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, n$. Assume that $v_i\left(t_0^-\right) \in V_i(0)$. Determine $j_i(k)$ and $q_i(k)$ for $i = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, n-1$ such that

$$j_i(k) \in C_i(k),\ q_i(k) \in Q_{i,j_i}(k)$$

$$\sum_{i=1}^{m} q_i(k) \in Q_{tgt}(k)$$

$$v_i(t_{k+1}) \in V_i(k+1)$$

and as "soft" constraints

$$\tau_{\mathrm{e},i}\left(t_{k+1}\right) \leq \tau_{\mathrm{e,max},i}$$

$$\tau_i\left(t_k\right) < \tau_i\left(t_{k+1}\right) \text{ or } \tau_i\left(t_k\right) \geq \tau_{\min,i}$$

where for $t_k < t \leq t_{k+1}$

$$v_i\left(t\right) = v_i\left(t_k\right) + \left(t - t_k\right)\left(q_{\mathrm{in},i}\left(k\right) - q_i\left(k\right)\right)$$

$$\tau_i\left(t\right) = \begin{cases} \left(t - t_k\right) & : j_i\left(k\right) \neq j_i\left(t_k^-\right) \\ \tau_i\left(t_k\right) + \left(t - t_k\right) & : j_i\left(k\right) = j_i\left(t_k^-\right) \end{cases}$$

$$\tau_{\mathrm{e},i}\left(t\right) = \begin{cases} 0 & : v_i\left(t\right) \leq v_{\mathrm{e},i} \\ t - t_{\text{last empty}} & : v_i\left(t\right) \leq v_{\mathrm{e},i} \text{ and } v_i\left(t\right) > v_{\mathrm{e},i} \\ \tau_i\left(t_k\right) + \left(t - t_k\right) & : v_i\left(t_k\right) > v_{\mathrm{e},i} \text{ and } v_i\left(t\right) > v_{\mathrm{e},i} \end{cases}$$

So we have $2 \times (5 + 1)$ inequalities per time step (2 for total discharge and 2 per pumping station for storage) and 5 variables (the discharges) per time step. These variables might be either integer or continuous or continuous with gaps in the allowed value range. In addition we have the soft constraints on run times and on emptying the system. For a simplified problem without run time or emptying constraints with on/off pumps, a 15 minute time step and 24 hour look-ahead we get 1152 inequalities for 480 binary variables.

4 Typical input data

In the Garmerwolde case there are five pumping stations. The inflow exhibits a roughly periodic sinusoidal pattern with a length of 24 hours with a minimum around 6 AM. Time step length is 15 minutes. Minima and maxima for the inflow patterns are given in Table 1. Note that the dry weather hourly inflows

Pumping station	min m^3/h	max m^3/h
Groningen (GR)	454	1179
Selwerd (SE)	126	423
G. Huizinga (GH)	144	600
Haren W. (HW)	80	155
Lewenborg (LE)	2	335

Table 1. Minima and maxima of hourly inflow

for all but Groningen are always below the minimum pumping capacity. Tow examples for the V and Q intervals are given in Table 2. A typical target flow range would be $Q_{\mathrm{tgt}} = [1700, 2200]$. Even with $O\left(10\right)$ possible combinations of pumps per time step, the number of possible selections in the unconstrained

Pumping station	Volumes (m^3)			Pumps (m^3/h)	
	$v_{e,i}$	Example 1	Example 2	Example 1	Example 2
Groningen (GR)	1000	[95,3957]	[95,6800]	[1800,2200]	[1000,4250]
Selwerd (SE)	121	[38,296]	[38,3200]]	[1620,1980]	[1060,2000]
G. Huizinga (GH)	176	[150,300]	[150,4900]	[330,400]	[1600,3280]
Haren W. (HW)	27	[22,50]	[22,1378]	[1020,1320]	[300,1200]
Lewenborg (LE)	30	[9,239]	[9,4100]	[1440,1760]	[550,1900]

Table 2. Data used in program

problem with 15 minute time step and 24 hour look ahead is $O\left(10^{96}\right)$. For the problem with variable flows there is the added complexity of selecting the flow for each pump. If the pumps were simple on/off pumps then fitting them into the range of target flows would be remarkably like the two dimensional cutting stock problem [1]. The problem also has points in common with the problem addressed in [4], but that approach does not allow for variable part supply rates. A result on the existence of solutions for a very special case is discussed in [3].

5 A simple greedy algorithm

We do a depth first search for a path of length n in a tree where the nodes correspond to system states at times t_k and the edges correspond to choices of flow ranges. At each node we generate all possible combinations of pump flow states, filter out those that are certain to lead to volume or target flow constraint violations, order the remainder so that we will first try those, that respect both "minimum run time" and "time since empty" constraints, then those that respect only "minimum run time", followed by those that respect only "time since empty" and finally those that do not respect either of these constraints. Within each group the flow state combinations where the pumps for districts furthest from empty are active are taken first. The first path to reach the horizon is used. For the selection of flow ranges in the first step along the path specific discharges are calculated. The main purpose of the search is to cope with the periodic variation in the inflow. Once a path is found the selection of flow ranges for the first step needs to be translated into actual pump flows.

5.1 Implementation with interval arithmetic

For narrow pump ranges Q and a narrow target range it makes sense to work in interval arithmetic for the system state. For wide ranges further investigations are needed. For narrow ranges we proceed as follows.

$$V_{\mathrm{fc},i}\left(t_0\right) = v_i\left(t_0^-\right)$$

A time step is processed as follows. For each $\boldsymbol{j} \in C_1(k) \times C_2(k) \times \cdots \times C_m(k)$ we apply the following accept/reject process. We calculate

$$Q''_{j_i}(k) = \left(\frac{V_i(k+1) - V_{\mathrm{fc},i}(t_k)}{\Delta_k} \right) \cap Q_{j_i}(k)$$

$$Q'_{j_i}(k) = Q''_{j_i}(k) \cap \left(Q_{\mathrm{tgt}}(k) - \sum_{u=1, u \neq i}^{m} Q_{j_u}(k) \right)$$

and then test

1. $\left(\sum_{i=1}^{m} Q'_{j_i}(k)\right) \cap Q_{\mathrm{tgt}}(k) \neq \emptyset$
2. $\left(V_{\mathrm{fc},i}(t_k) + \Delta_k Q'_{j_i}(k)\right) \cap V_i(k+1) \neq \emptyset$

If both tests then we define

$$V_{\mathrm{fc},i}(t) = \left(V_{\mathrm{fc},i}(t_k) + (t - t_k) Q'_{j_i}(k)\right)$$

$$V_{\mathrm{fc},i}(t_{k+1}) = \left(V_{\mathrm{fc},i}(t_k) + (t - t_k) Q'_{j_i}(k)\right) \cap V_i(k+1)$$

Next we calculate $\tau_i(t)$ as before, but we set

$$\tau_{\mathrm{e},i}(t) = \begin{cases} 0 & : v_{\mathrm{e},i} \in V_{\mathrm{fc},i}(t) \\ t - t_{\text{last empty}} & : v_{\mathrm{e},i} \in V_{\mathrm{fc},i}(t_k) \text{ and } V_{\mathrm{fc},i}(t) > v_{\mathrm{e},i} \\ \tau_i(t_k) + (t - t_k) & : V_{\mathrm{fc},i}(t_k) > v_{\mathrm{e},i} \text{ and } V_{\mathrm{fc},i}(t) > v_{\mathrm{e},i} \end{cases}$$

6 Plans

First trials with the algorithm showed a potential problem with sedimentation due to long stays of sewage in the urban sewer system. Probably a slowly varying target flow (changes of up to $100\mathrm{m}^3/\mathrm{h}$ per hour will not adversely affect the WWTP) would be a better option. Interval arithmetic is probably desirable as we are constantly checking constraints on calculated values, but it might be a good idea to preselect much narrower intervals for the calculation of $V_{\mathrm{fc},i}(t_{k+1})$ from $V_{\mathrm{fc},i}(t_k)$.

References

1. Dyckhoff, H.: A typology of cutting and packing problems. European Journal of Operational Research 44(2), 14–159 (Jan 1990)
2. NLingenieurs Sewer Systems Workgroup: Sewer Systems Module for Higher Professional Education. KIVI-NIRIA, The Hague, The Netherlands (2009), version 2 (Translated by M.C. de Geus)
3. van Nooijen, R.R., Kolechkina, A.G.: Realizing steady supply to a treatment plant from multiple sources. In: 6th IFAC Symposium on System Structure and Control, 22 to 24 June, Istanbul (2016), to be published.
4. Perkins, J.R., Kumark, P.R.: Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems. IEEE Transactions on Automatic Control 34(2), 139–148 (Feb 1989)

Collective Defense and Possible Relaxations in Weighted Abstract Argumentation Problems

Stefano Bistarelli

Dep. of Mathematics and Computer Science
University of Perugia
Via Vanvitelli, 1 - 06123 Perugia, Italy
bista@dmi.unipg.it

Abstract. When dealing with Weighted Abstract Argumentation, having weights on attacks clearly brings more information. The advantage, for instance, is the possibility to define a different notion of defense, checking also if the weight associated with defense is stronger than the weight of attack. In this work, we study two different relaxations, one related to a new weighted defense we propose, by checking the difference between the composition of inward and outward attack- weights. The second one is related to how much inconsistency we are willing to tolerate inside an extension; such amount is computed by aggregating the costs of the attacks between any two arguments both inside an extension. These two relaxations are strictly linked: allowing a small conflict may lead to have more arguments into an extension, and consequently result in a stronger or weaker defense. We deal with weights with a semiring structure, which can be instantiated to different metrics used in the literature (e.g., fuzzy).

How Neural Networks (NN) Can (Hopefully) Learn Faster by Taking Into Account Known Constraints

Chitta Baral[1], Martine Ceberio[2], and Vladik Kreinovich[2]

[1] Department of Computer Science, Arizona State University
Tempe, AZ 85287-5406, USA, chitta@asu.edu
[2] Department of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA, mceberio@utep.edu, vladik@utep.edu

Abstract. Neural networks are a very successful machine learning technique. At present, deep (multi-layer) neural networks are the most successful among the known machine learning techniques. However, they still have some limitations, One of their main limitations is that their learning process still too slow. The major reason why learning in neural networks is slow is that neural networks are currently unable to take prior knowledge into account. As a result, they simple ignore this knowledge and simulate learning "from scratch". In this paper, we show how neural networks can take prior knowledge into account and thus, hopefully, learn faster.

1 Formulation of the Problem

Need for maching learning. In many practical situations, we know that the quantities $y_1, \ldots, y_L$ depend on the quantities $x_1, \ldots, x_n$, but we do not know the exact formula for this dependence. To get this formula, we measure the values of all these quantities in different situations $m = 1, \ldots, M$, and then use the corresponding measurement results $x_i^{(m)}$ and $y_\ell^{(m)}$ to find the corresponding dependence. Algorithms that "learn" the dependence from the measurement results are known as *machine learning* algorithms.

Neural networks (NN): main idea and successes. One of the most widely used machine learning techniques is the technique of *neural networks* (NN) – which is based on a (simplified) simulation of how actual neurons works in the human brain (a brief technical description of this technique is given in Section 2). This technique has many useful applications; see, e.g., [1, 2].

At present (2016) multi-layer ("deep") neural networks are, empirically, the most efficient of the known machine learning techniques.

Neural networks: limitations. One of the main limitations of neural networks is that their learning very slow: they need many thousand iterations just to learn a simple dependence.

This slowness is easy to explain: the current neural networks always start "from scratch", from zero knowledge. In terms of simulating human brain, they do not simulate how we learn the corresponding dependence – they simulate how a newborn child will eventually learn to recognize this dependence. Of course, this inability to take any prior knowledge into account drastically slows down the learning process.

What is prior knowledge. Prior knowledge means that we know some relations ("constraints") between the desired values $y_1, \ldots, y_L$ and the observed values $x_1, \ldots, x_n$, i.e., we know several relations of the type $f_c(x_1, \ldots, x_n, y_1, \ldots, y_L) = 0$, $1 \le c \le C$.

Prior knowledge helps humans learn faster. Prior knowledge helps us learn. Yes, it takes some time to learn this prior knowledge, but this has been done *before* we have samples of x_i and y_ℓ. As a result, the time from gathering the samples to generating the desired dependence decreases.

This is not simply a matter of accounting: the same prior knowledge can be used (and usually is used) in learning several different dependencies. For example, our knowledge of sines, logarithms, of calculus helps in finding the proper dependence in many different situations. So, when we learn the prior knowledge first, we decrease the overall time needed to learn all these dependencies.

How to speed up artificial neural networks: a natural idea. In view of the above explanation, a natural idea is to enable neural networks to take prior knowledge into account. In other words, instead of learning all the data "from scratch", we should first learn the constraints. Then, when it is time to use the data, we should be able to use these constraints to "guide" the neural network in the right direction.

What we do in this paper. In this paper, we show how to implement this idea and thus, how to (hopefully) achieve the corresponding speed-up.

To describe this idea, we first, in Section 2, recall how the usual NN works. Then, in Section 3, we show how we can perform a preliminary training of a NN, so that it can learn to satisfy the given constraints. Finally, in Section 4, we show how to train the resulting pre-trained NN in such a way that the constraints remain satisfied.

2 Neural Networks: A Brief Reminder

Signals in a biological neural network. In a biological neural network, a signal is represented by a sequence of spikes. All these spikes are largely the same, what is different is how frequently the spikes come.

Several sensor cells generate such sequences: e.g., there are cells that translate the optical signal into spikes, there are cells that translate the acoustic signal into spikes. For all such cells, the more intense the original physical signal, the more spikes per unit time it generates. Thus, the frequency of the spikes can serve as a measure of the strength of the original signal.

From this viewpoint, at each point in a biological neural network, at each moment of time, the signal can be described by a single number: namely, by the frequency of the corresponding spikes.

What is a biological neuron: a brief description. A biological neuron has several inputs and one output. Usually, spikes from different inputs simply get together – probably after some filtering. Filtering means that we suppress a certain proportion of spikes. If we start with an input signal x_i, then, after such a filtering, we get a decreased signal $w_i \cdot x_i$. Once all the inputs signals are combined, we have the resulting signal $\sum\limits_{i=1}^{n} w_i \cdot x_i$.

A biological neuron usually has some excitation level w_0, so that if the overall input signal is below w_0, there is practically no output. The intensity of the output signal thus depends on the difference $d \stackrel{\text{def}}{=} \sum\limits_{i=1}^{n} w_i \cdot x_i - w_0$. Some neurons are linear, their output is proportional to this difference. Other neurons are non-linear, they output is equal to $s_0(d)$ for some non-linear function $s_0(z)$. Empirically, it was found that the corresponding non-linear transformation takes the form $s_0(z) = 1/(1 + \exp(-z))$.

Comment. It should be mentioned that this is a simplified description of a biological neuron: the actual neuron is a complex *dynamical* system, in the sense that its output at a given moment of time depends not only on the current inputs, but also on the previous input values.

Artificial neural networks and how they learn. If we need to predict the values of several outputs $y_1, \ldots, y_\ell, \ldots, y_L$, then for each output y_ℓ, we train a separate neural network.

In an artificial neural networks, input signals $x_1, \ldots, x_n$ first go to the neurons of the first layer, then the results go to neurons of the second layer, etc.

In the simplest (and most widely used) arrangement, the second layer has linear neurons. In this arrangement, the neurons from the first layer produce the signals $y_{\ell,k} = s_0\left(\sum\limits_{i=1}^{n} w_{\ell,ki} \cdot x_i - w_{\ell,k0}\right)$, $1 \le k \le K_\ell$, which are then combined into an output $y_\ell = \sum\limits_{k=1}^{K} W_{\ell,k} \cdot y_k - W_{\ell,0}$. This is called *forward propagation.* (In this paper, we will only describe formulas for this arrangement, since formulas for the multi-layer neural networks can be obtained by using the same idea.)

How a NN learns: derivation of the formulas. Once we have an observation $(x_1^{(m)}, \ldots, x_n^{(m)}, y_\ell^{(m)})$, we first input the values $x_1^{(m)}, \ldots, x_n^{(m)}$ into the current NN; the network generates some output $y_{\ell,NN}$. In general, this output is different from the observed output $y_\ell^{(m)}$. We therefore want to modify the weights $W_{\ell,k}$ and $w_{\ell,ki}$ so as to minimize the squared difference $J \stackrel{\text{def}}{=} (\Delta y_\ell)^2$, where $\Delta y_\ell \stackrel{\text{def}}{=} y_{\ell,NN} - y_\ell^{(m)}$. This minimization is done by using gradient descent, where each of the unknown values is updated as $W_{\ell,k} \to W_{\ell,k} - \lambda \cdot \dfrac{\partial J}{\partial W_{\ell,k}}$ and $w_{\ell,ki} \to$

$w_{\ell,ki} - \lambda \cdot \dfrac{\partial J}{\partial w_{\ell,ki}}$. The resulting algorithm for updating the weights is known as *backpropagation*. This algorithm is based on the following idea.

First, one can easily check that $\dfrac{\partial J}{\partial W_{\ell,0}} = -2\Delta y$, so $\Delta W_{\ell,0} = -\lambda \cdot \dfrac{\partial J}{\partial W_{\ell,0}} = \alpha \cdot \Delta y_\ell$, where $\alpha \stackrel{\text{def}}{=} 2\lambda$. Similarly, $\dfrac{\partial J}{\partial W_{\ell,k}} = 2\Delta y_\ell \cdot y_{\ell,k}$, so $\Delta W_{\ell,k} = -\lambda \cdot \dfrac{\partial J}{\partial W_{\ell,k}} = 2\lambda \cdot \Delta y_\ell \cdot y_{\ell,k}$, i.e., $\Delta W_{\ell,k} = -\Delta W_{\ell,0} \cdot y_{\ell,k}$.

The only dependence of y_ℓ on $w_{\ell,ki}$ is via the dependence of $y_{\ell,k}$ on $w_{\ell,ki}$. So, for $w_{\ell,k0}$, we can use the chain rule and get $\dfrac{\partial J}{\partial w_{\ell,k0}} = \dfrac{\partial J}{\partial y_{\ell,k}} \cdot \dfrac{\partial y_{\ell,k}}{\partial w_{\ell,k0}}$, hence:

$$\frac{\partial J}{\partial w_{\ell,k0}} = 2\Delta y_\ell \cdot W_{\ell,k} \cdot s_0'\left(\sum_{i=1}^{n} w_{\ell,ki} \cdot x_i - w_{\ell,k0}\right) \cdot (-1).$$

For $s_0(z) = 1/(1+\exp(-z))$, we have $s_0'(z) = \exp(-z)/(1+\exp(-z))^2$, i.e.,

$$s_0'(z) = \frac{\exp(-z)}{1+\exp(-z)} \cdot \frac{1}{1+\exp(-z)} = s_0(z) \cdot (1 - s_0(z)).$$

Thus, in the above formula, where $s_0(z) = y_{\ell,k}$, we get $s_0'(z) = y_{\ell,k} \cdot (1 - y_{\ell,k})$, $\dfrac{\partial J}{\partial w_{\ell,k0}} = -2\Delta y_\ell \cdot W_{\ell,k} \cdot y_{\ell,k} \cdot (1 - y_{\ell,k})$, and

$$\Delta w_{\ell,k0} = -\lambda \cdot \frac{\partial J}{\partial w_{\ell,k0}} = \lambda \cdot 2\Delta y_\ell \cdot W_{\ell,k} \cdot y_{\ell,k} \cdot (1 - y_{\ell,k}).$$

So, we have $\Delta w_{\ell,k0} = -\Delta W_{\ell,k} \cdot W_{\ell,k} \cdot (1 - y_{\ell,k})$. For $w_{\ell,ki}$, we have

$$\frac{\partial J}{\partial w_{\ell,ki}} = 2\Delta y_\ell \cdot W_{\ell,k} \cdot y_{\ell,k} \cdot (1 - y_{\ell,k}) \cdot x_i = -\frac{\partial J}{\partial w_{\ell,k0}} \cdot x_i,$$

hence $\Delta w_{\ell,ki} = -x_i \cdot \Delta w_{\ell,k0}$. Thus, we arrive at the following algorithm:

Resulting algorithm. We pick some value α, and cycle through observations $(x_1, \ldots, x_n)$ with the desired outputs y_ℓ. For each observation, we first apply the forward propagation to compute the network's prediction $y_{\ell,NN}$, then we compute $\Delta y_\ell = y_{\ell,NN} - y_\ell$, $\Delta W_{\ell,0} = \alpha \cdot \Delta y_\ell$, $\Delta W_{\ell,k} = -\Delta W_{\ell,0} \cdot y_{\ell,k}$, $\Delta w_{\ell,k0} = -\Delta W_{\ell,k} \cdot W_{\ell,k} \cdot (1 - y_{\ell,k})$, and $\Delta w_{\ell,ki} = -\Delta w_{\ell,k0} \cdot x_i$, and update each weight w to $w_{\text{new}} = w + \Delta w$. We repeat this procedure until the process converges.

3 How to Pre-Train a NN to Satisfies Given Constraints

To train the network, we can use any observations $(x_1^{(m)}, \ldots, x_n^{(m)}, y_1^{(m)}, \ldots, y_L^{(m)})$ that satisfy all the known constraints.

To satisfy the constraints $f_c(x_1, \ldots, x_n, y_1, \ldots, y_L) = 0$, $1 \le c \le C$, means to minimize the distance from the vector of values $(f_1, \ldots, f_C)$ to

the ideal point $(0, \ldots, 0)$, i.e., equivalently, to minimize the sum $F \stackrel{\text{def}}{=} \sum_{c=1}^{C} (f_c(x_1, \ldots, x_n, y_1, \ldots, y_L))^2$. To minimize this sum, we can use a similar gradient descent idea. From the mathematical viewpoint, the only difference from the usual backpropagation is the first step: here,

$$\frac{\partial F}{\partial W_{\ell,0}} = 2 \cdot \sum_{c=1}^{C} f_c \cdot \frac{\partial f_c}{\partial y_\ell}, \quad \text{hence} \quad \Delta W_{\ell,0} = -\alpha \cdot \sum_{c=1}^{C} f_c \cdot \frac{\partial f_c}{\partial y_\ell}.$$

Once we have computed $\Delta W_{\ell,0}$, all the other changes $\Delta W_{\ell,k}$ and $\Delta w_{\ell,ki}$ are computed based on the same formulas as above.

The consequence of this algorithm modification is that instead of L independent neural networks used to train each of the L outputs y_ℓ, now we have L dependent ones. The dependence comes from the fact that, to start a new cycle for each ℓ, we need to know the values $y_1, \ldots, y_L$ corresponding to all the outputs.

4 How to Retain Constraints When Training Neural Networks on Real Data

Once the networks is pre-trained so that the constraints are all satisfied, we need to train it on the real data. In this real-data training, we need to make sure that not only all the given data points fit, but that also all C constraints remain satisfied. In other words, on each step, we need to make sure not only that Δy_ℓ is close to 0, but also that $f_c(x_1, \ldots, x_n, y_1, \ldots, y_L)$ is close to 0 for all ℓ. So, similar to the previous section, instead of minimizing $J = (\Delta y_\ell)^2$, we should minimize a combined objective function $G \stackrel{\text{def}}{=} J + N \cdot F$, where N is an appropriate constant, and $F = \sum_{c=1}^{C} f_c^2$.

Similarly to pre-training, the only difference from the usual backpropagation algorithm is that we compute the values $\Delta W_{\ell,0}$ differently:

$$\Delta W_{\ell,0} = \alpha \cdot \left(\Delta y_\ell - N \cdot \sum_{c=1}^{C} f_c \cdot \frac{\partial f_c}{\partial y_\ell} \right).$$

Acknowledgments. This work was supported in part by NSF grants HRD-0734825, HRD-1242122, and DUE-0926721, and by an award from Prudential Foundation.

References

1. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, N.Y., 2006.
2. G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets", *Neural Computation*, 2006, Vol. 18, pp. 1527–1554.

Construction of a Mosaic from an Underwater Video: an Interval Analysis Approach

M. Laranjeira, L. Jaulin, S. Tauvry, and C. Aubry

ENSTA-Bretagne, Office M024
2 rue Franois Verny, 29806 Brest, France
lucjaulin@gmail.com

Design of PI Controllers based on Constraint Programming using Frequency Envelope Condition

Harsh Purohit and P.S.V.Nataraj

Systems and Control Engineering Group
Indian Institute of Technology Bombay,Powai-400076, Mumbai
harsh.purohit@sc.iitb.ac.in,
nataraj@sc.iitb.ac.in

Abstract. This article presents a new design method for PI controller based on constraint programming. The design is developed to reject the load disturbance with constraint on sensitivity. It also includes the frequency envelope condition as an additional constraint which avoids sampling in the frequency interval. Interval constraint solver, RealPaver, is used to solve the formulated constraint satisfaction problem.

Keywords: PI Controller, Constraint Satisfaction Problem

1 Introduction

It is well-known that PI/D type controllers are dominating the control and process industry. The main reasons are its simplicity, performance robustness and many effective tuning methods based on minimum system model knowledge. Ziegler and Nichols [1] is one of the simplest and useful tuning method, but it is not applicable to a wide range of systems.

In the last decade, many researchers have developed different tuning methods based on frequency domain method, interval model control method and optimization method [2, ref therein]. In spite of the wide spread use of PI/D controllers there is a lack of a universally accepted tuning method. The present work can be viewed as a simple tuning method based on constraint satisfaction problem.

2 The proposed method

The block diagram of close loop control system is illustrated in Figure 1. $P(s)$ is process transfer function and $G_c(s)$ is PI controller transfer function. The process output, set point and control signals are described as y_{sp}, y and u respectively. The main design objective is to determine the controller parameters (k_p,k_i) so that the system behaves well with respect to the external disturbance.

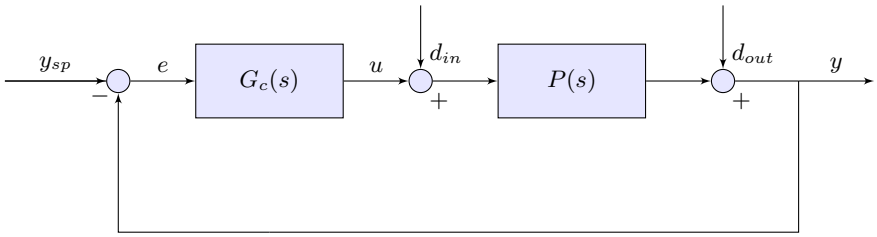


Fig. 1. Block diagram of the system with input and output disturbance

The relationship between d_{in}, d_{out} and process output y can be described as follows

$$Y(s) = \frac{P(s)}{1 + G_c(s)P(s)} D_{in}(s) + \frac{1}{1 + G_c(s)P(s)} D_{out}(s)$$

We then define

$$S(s) = \frac{1}{1 + G_c(s)G(s)}$$

$S(s)$ is called the sensitivity function and it determines the robustness to unmodeled system dynamics and rejection to external disturbance. The peak value of the sensitivity function are denoted by

$$M_s = \max_{0<\omega<\infty} |S(j\omega)|$$

It has been observed that in the real world disturbance signals are generally in the low frequency range. Therefore, by choosing the appropriate value of M_s (1.2-2) over a design frequency range we can get constraint on the sensitivity of the system. According to Astrom *et al.* [3] maximizing integral gain k_i is equivalent to minimizing the integral error (IE) for a step change in the load disturbance.

Furthermore, sensitivity constraint can be represented as follows

$$\psi(k_p, k_i, \omega) := \left|1 + (k_p - j\frac{k_i}{\omega})G(j\omega)\right|^{-1}$$

$$\psi(k_p, k_i, \omega) \leq M_s \qquad \forall \omega \in \tilde{\omega}$$

where $\tilde{\omega} = [\omega_l, \omega_h]$ is design frequency interval. We can express $G(j\omega) = \alpha(\omega) + j\beta(\omega)$. By simple algebraic manipulation, this constraint can be simplified as follows

$$\psi(k_p, k_i, \omega) = 1 + 2\alpha k_p + r^2 k_p^2 + 2\frac{\beta}{\omega} k_i + \frac{r^2}{\omega^2} k_i^2 \tag{1}$$

where $r(\omega)^2 = \alpha^2(\omega) + \beta^2(\omega)$ and we have dropped ω argument in (1) for notation simplicity. At fix frequency, this constraint represent the ellipse in $k_p - k_i$ plane. For example, see Figure 2. Therefore, we get an envelope for all frequency $0 \leq \omega \leq \infty$.

$$\frac{\partial \psi}{\partial \omega}(k_p, k_i, \omega) = 0$$

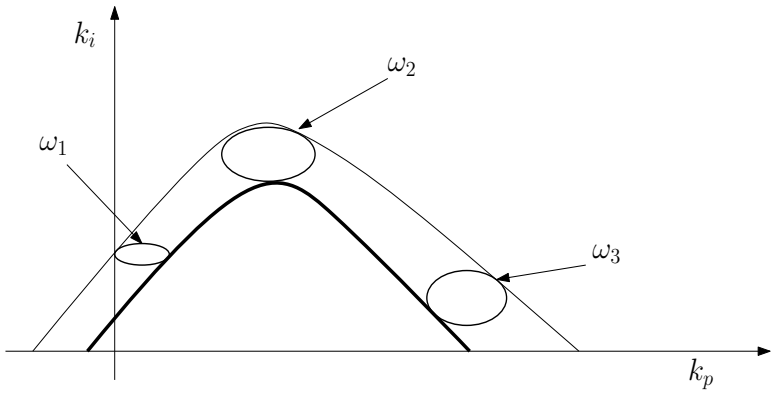


Fig. 2. Geometrical illustration of the sensitivity constraint and the envelope generated by it.(dark solid line)

To find the maximum of this function,

$$d\psi = \frac{\partial \psi}{\partial k_p} dk_p + \frac{\partial \psi}{\partial k_i} dk_i + \frac{\partial \psi}{\partial \omega} d\omega = 0$$

Hence, at the extremum $dk_i = 0$, for arbitrary variation dk we must have $\frac{\partial \psi}{\partial k_p} = 0$. Combining this with envelope condition we get

$$\frac{\partial \psi}{\partial k_p}(k_p, k_i, \omega) = 2a + 2r^2 k_p = 0$$

$$\frac{\partial \psi}{\partial \omega}(k_p, k_i, \omega) = 2\Big(\frac{b}{\omega}\Big)' k_i + \Big(\frac{r^2}{\omega^2}\Big)' k_i^2 + sa' k_p + 2rr' k_p^2 = 0 \tag{2}$$

$$\psi(k_p, k_i, \omega) = M_s$$

where prime represents differentiation with respect to ω. For imposing the stability, we have to add two more constraints

$$k_p \beta - k_i \frac{\alpha}{\omega} = 0, \qquad k_p \alpha + k_i \frac{\beta}{\omega} = -1 \tag{3}$$

Thus, the design problem is reduced to solving nonlinear algebraic equations (2) and (3). For solving, we can consider this problem as a constraint satisfaction problem (CSP). Given a vector $(x_1, x_2 \dots x_n) \in \mathbb{R}^n$ of unknown a constraint system $(\mathcal{C}, X)$ is defined by a set $\mathcal{C} = c_1 \dots c_p$ of constraints and a bounded domain $X = x_1 \times \dots \times x_n$.

$$\begin{cases} c_i : f_i(x_1 \dots x_n) = 0, & i = 1, \dots, m, \\ x_k = \{r \in \mathbb{R} | a_k \le r \le b_k\} & k = 1, \dots, n. \end{cases}$$

In this work, we have used software package RealPaver [4] which implements a modeling language and interval-based algorithms to process systems of nonlinear constraints over the real numbers. Given a CSP, RealPaver computes a union of boxes that contains all the solutions. Among all the solution boxes we choose the solution box with maximum value of k_i as discussed earlier. Different consistency techniques like box, hull and 3B has been implemented in RealPaver. A consistency property determines which values from domains are consistent or not with respect to the constraint of CSP.

3 The Design Procedure

According to previous section PI controller can be designed using following steps.

Step:1 Derive the nonlinear algebraic constraints (2) and (3) for sensitivity specification M_s with unknowns k_p, k_i and ω.

Step:2 Model the overall problem as an constraint satisfaction problem and solve with appropriate consistency technique in RealPaver.

Step:3 If the solution does not exist then repeat step:1 with different specification. If we get the solution boxes then choose the solution with maximum k_i value.

4 Conclusion

This article describes a design method for PI controller. The primary goal is to obtain load disturbance rejection. This is done by constraining maximum sensitivity M_s. The design problem can be transformed as a constrained satisfaction problem with nonlinear algebraic equations. We have developed a procedure to design and solve the CSP numerically with RealPaver. The method will give a solution with less time and good precision. It is also helpful to indicate when there is no PI controller exist for given specifications.

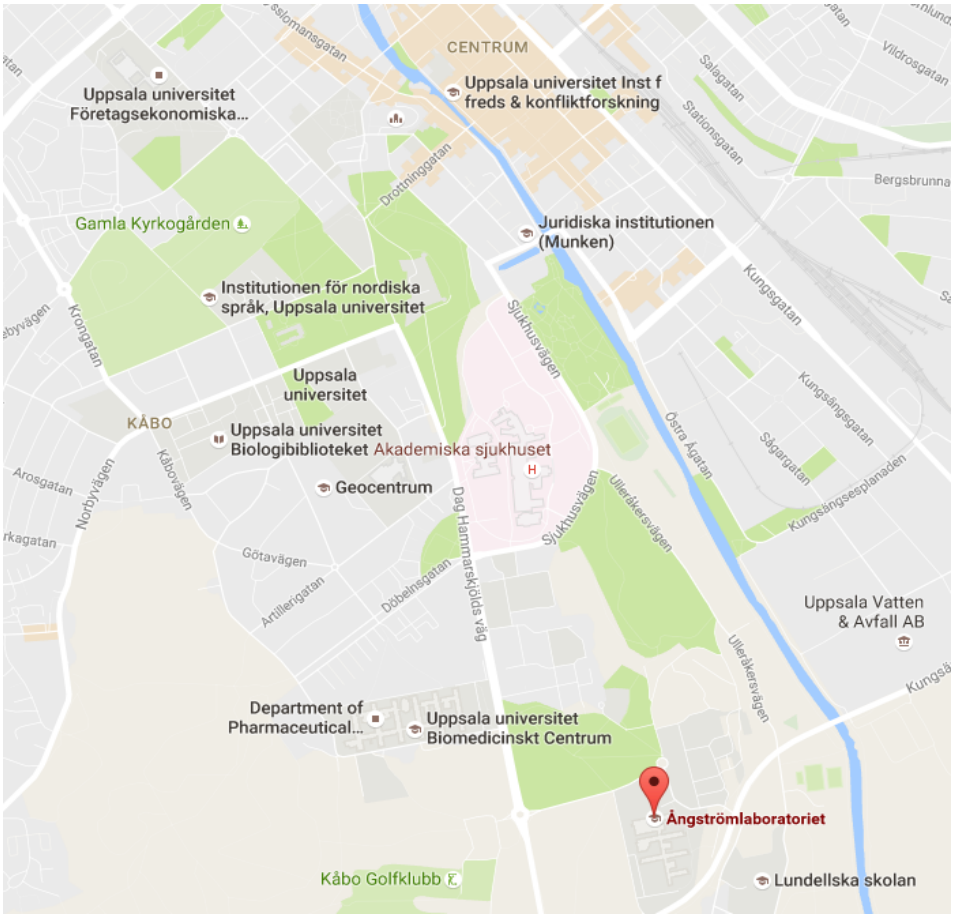
References

1. Ziegler J. G., N. B. Nichols: Optimum settings for automatic controllers. Trans. ASME. 67, 759–768 (1942)
2. Ch. Anil, R.Padma Sree: Tuning of PID controllers for integrating systems using direct synthesis method. ISA Transactions.57,211–219 (2015)
3. Astrom K.J., H. Panagopoulos, T. Hagglund: Design of PI controllers based on non-convex optimization. Automatica.35:5 (1998)
4. L. Granvilliers, F. Benhamou: Algorithm 852: Realpaver: an Interval Solver using Constraint Satisfaction Techniques. ACM TOMS.32(1) 138–156 (2006)

UPPSALA UNIVERSITET

THE UNIVERSITY OF TEXAS AT EL PASO

Directions



The lectures will take place at
the Ångströmlaboratoriet

Located at:
752 37 Uppsala, Sweden

Organized By

Martine Ceberio & Vladik Kreinovich
The University of Texas At El Paso