If We Take Into Account that Constraints Are Soft, Then Processing Constraints Becomes Algorithmically Solvable

Quentin Brefort and Luc Jaulin
ENSTA-Bretagne, LabSTICC, IHSEV, OSM
2 rue François Verny, 29806 Brest, France
Quentin.Brefort@ensta-bretagne.org, luc.jaulin@ensta-bretagne.fr

Martine Ceberio and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, Texas 79968, USA
mceberio@utep.edu, vladik@utep.edu

***Abstract*—Constraints are ubiquitous in science and engineering. Constraints describe the available information about the state of the system, constraints describe possible relation between current and future states of the system, constraints describe which future states we would like to obtain. To solve problems from engineering and science, it is therefore necessary to process constraints. We show that if we treat constraints as *hard* (crisp), with all the threshold values exactly known, then in the general case, all the corresponding computational problems become algorithmically unsolvable. However, these problems become algorithmically solvable if we take into account that in reality, constraints are *soft*: we do not know the exact values of the corresponding thresholds, we do not know the exact dependence between the present and future states, etc.**

I. Formulation of the Problem: Constraint Processing Is Ubiquitous in Science and Engineering

Main objectives of science and engineering: reminder. The main objective of science is to describe the world. The main objective of engineering is to predict consequences of different actions, different designs – and to select actions and designs which will lead to the future state with desired properties.

Main objectives of science and engineering: towards a precise description. Our information about the physical world usually comes in terms of the numerical values of different physical quantities. To describe the weather, we describe the temperature, the atmospheric pressure, the components of wind velocity, etc. To describe the health of a patient, we list numerous numbers coming from the blood test, from – if needed – EKG, etc.

The actual state of the world is therefore described in terms of the values of the corresponding physical quantities $x_1, \ldots, x_n$. In these terms, to describe the world means to find out as much as possible about the possible values of the corresponding tuple $x = (x_1, \ldots, x_n)$.

To understand the consequences of a certain action means to find out as much as possible about the future values $y_1, \ldots, y_m$ of the relevant quantities – and we should be able to find the action for which the future state satisfies the desired properties.

Let us show that constraints are ubiquitous. Let us show that a natural description of all these objective necessitates the use of constraints.

Constraints are important in determining the state of the world. Let us start with the very first objective: determining the state of the world. Our information about the world comes from measurements. An ideal measurement of a physical quantity returns the exact value of this quantity. However, in practice, measurements are never ideal:

- Measurement never result in a *single* value of a quantity.
- Instead, measurements result in describing a *set* of possible values of the corresponding quantity.

For example, if:

- after measuring a quantity x_i, we get the value $\widetilde{x}_i$, and
- the manufacturer of the measuring instrument provided us with an upper bound Δ_i on the corresponding measurement error $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ ($|\Delta x_i| \le \Delta_i$),

then the only information that we gain about the actual (unknown) value x_i of this quantity is that this value belongs to the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$; see, e.g., [14].

Some measuring instruments measure one of the basic quantities x_i; other measuring instruments measure some *combinations* $y = f(x_1, \ldots, x_n)$ of these quantities. In this case, after we perform the measurement and find out that the value y is between the corresponding bounds $\underline{y} \stackrel{\text{def}}{=} \widetilde{y} - \Delta$ and $\overline{y} \stackrel{\text{def}}{=} \widetilde{y} + \Delta$, we can then conclude that the actual (unknown) tuple belongs to a set

$$\{(x_1, \ldots, x_n) : \underline{y} \le f(x_1, \ldots, x_n) \le \overline{y}\}. \qquad (1)$$

From the computational viewpoint, a set is a *constraint*; see, e.g., [2], [3], [5], [11], [15]. Thus, constraints are important in determining the state of the world.

Constraints are important in predicting the future states. To make predictions about the future state $y = (y_1, \ldots, y_m)$ based on the current state x, we need to know the relation between x and y.

In simple situations, the corresponding relation is straightforward: the available information about the state x enables us to uniquely determine the state y. For example, this is the case of simple mechanical systems: once we knowing the initial location and velocity of a plant, we can uniquely predict its position at future moments of time.

In other cases, however, the future state y depends not only on the current information about the state, but also on many unknown factors. For example, even if we know the exact weather conditions in an area, this is not enough to make a long-term weather prediction for this area: future weather is also affected by difficult-to-measure factors such as the in-depth oceanic behavior.

In general, we do not have a deterministic dependence of y on x; instead, we have a *relation*, i.e., the set $R \subseteq X \times Y$ of possible pairs (x, y).

In computational terms, a relation is a *constraint*. Thus, constraints are important for predicting consequences of different actions.

Constraints are important for selecting an action. Let $a = (a_1, \ldots, a_p)$ be parameters describing possible actions, and let $A \subseteq \mathbb{R}^p$ be a set of possible actions.

In the *ideal* case:

- we know the current state x,
- we know how different actions will affect this state, i.e., we know the relation $y = f(x, a)$, and
- we have a constraint that the future state y must satisfy.

For example, when we design a car engine, we must make sure that its energy efficiency y_1 exceeds the required threshold, that the concentration y_2 of potential pollutants in its exhaust does not exceed a certain level, etc. If we describe the corresponding set of desired tuples y by D, then the problem is to find the set $R = \{a \in A : f(x, a) \in D\}$ of actions which result in the satisfaction of all the required constraints.

In a *more realistic* case, instead of the exact state x, we only know a set S of possible states x. In this case, we need to find actions which would lead to the satisfaction of the desired constraints for all possible states x. In other words, we need to describe the state

$$\{a \in A : f(x, a) \in D \text{ for all } x \in S\}. \qquad (2)$$

Need for processing constraints. The above analysis shows that in solving problems from science and engineering, we need to process constraints. Let us list the resulting computational problems:

- First, we need to describe the results of each measurement. Specifically, once we have a computable function $f(x_1, \ldots, x_n)$ and computable values $\underline{y}$ and $\overline{y}$, we need to describe the set $\{x : \underline{y} \le f(x) \le \overline{y}\}$ of possible tuples x – i.e., of all the tuples x which are consistent with the results of this measurement.
- Second, we need to be able to combine the results of several measurements. In other words:
  - we know the set S_1 of all the tuples which are consistent with the first measurement;
  - we know the set S_2 of all the tuples which are consistent with the second measurement;
  - ...
  - we know the set S_m of all the tuples which are consistent with the m-th measurement;
  - we need to describe the set $S = S_1 \cap \ldots \cap S_m$ of all the tuples which are consistent with the results of all available measurements.
- Then, we need to be able to predict the future state:
  - we know the set $S \subseteq X$ of possible states of the world;
  - we know the relation $R \subseteq X \times Y$ that describes the system's dynamics;
  - we need to describe the set of possible Y of possible future states:

    $$Y = \{y : (x, y) \in R \text{ for some } x \in S\}. \qquad (3)$$

    In mathematical terms, Y is known as a *composition* $Y = R \circ S$.
- Finally, we need to describe the set of possible actions:
  - we know the set $S \subseteq X$ of possible states;
  - we know the set A of possible actions;
  - we know the set $D \subseteq Y$ of desired future states;
  - we know a computable function $f(x, a)$ that describes how the future state depends on the initial state x and on the action a;
  - we need to describe the state of actions that lead to the desired goal

    $$\{a \in A : f(x, a) \in D \text{ for all } x \in S\}.$$

What we do in this paper. In this paper, we start by describing the above four fundamental problems of constraint processing in precise algorithmic terms. We then show that in general, all these problems are *algorithmically unsolvable*.

At first glance, this sounds like one of these negative results that have been appearing starting with the 1930s Goedel's Theorem. However, we will show that for our constraint processing problems, the situation is not as negative as it may seem. To be more precise:

- The situation is indeed negative if we assume that all the constraint are known *exactly*: that we know the exact bounds on the measurement error, that we know the exact relation between the present and future states, etc.
- However, in reality, these constraints are only know *approximately*. In other words, the constraints are actually *soft* [2], [3], [5], [11], [15]: the corresponding numerical bounds are only approximately known, the resulting sets may somewhat deviate from their exact form, etc. We will then show that if we take this softness into account, then all four fundamental problems become algorithmically solvable.

II. What Is Computable, When Constraints Are Computable, etc.: A Reminder

Mathematical objects for which we need to describe what is computable and what is not. In the above analysis, we started with the simplest object – a *real number* – which describe the value of a single quantity. Next, we considered *tuples* of real numbers – which describe states. After that, we considered *functions* and *sets* (*constraints*). To analyze which problems related to these objects are algorithmically solvable and which are not, we must first describe these objects in precise algorithmic terms.

Comment. A detailed description of computable objects and their properties can be found in [1], [10], [13], [16].

What is a computable real number? Let us start our description of what is computable and what is not with the simplest object – a real number. In physical terms, a real number describes the actual value of a physical quantity. As we have mentioned earlier, it is not possible to learn the exact value of this quantity, measurements can only provide us with *approximate* values. From this viewpoint, it is reasonable to call a real number computable if we can algorithmically predict, for each measurement, what will be the corresponding measurement result.

Let us describe this idea in precise terms. Most modern measuring instruments produce the measurement result in the binary form: as a sequence of bits describing the value of the measured quantity. Each measuring instrument provides only an approximate value of the quantity; in other words, it provides only the few (k) first bits in the binary expansion of the actual (unknown) value x. In mathematical terms, the measurement result is thus a rational number r_k for which $|x - r_k| \le 2^{-k}$. Thus, we arrive at the following definition.

Definition 1. *A real number x is called* computable *if there exists an algorithm that, given a natural number k, returns a rational number r_k for which $|x - r_k| \le 2^{-k}$.*

What is a computable tuple? This definition is straightforward: a tuple $x = (x_1, \ldots, x_n)$ is computable if and only if all n real numbers $x_1, \ldots, x_n$ are computable.

All physical quantities are bounded: an observation. From the purely mathematical viewpoint, a real number can take any value from $-\infty$ to $+\infty$. In practice, however, for each physical quantity, we usually know the bounds. For example:

- in physical measurements, a velocity cannot exceed the speed of light;
- in meteorological measurements, temperature must be between $-80°$ and $+50°$ C, and there are also bounds on wind speed, velocity., etc.

In the following text, we will denote the known lower and upper bounds on the i-th quantity by, correspondingly, L_i and U_i; then, all physically possible values x_i must be from the interval $[L_i, U_i]$.

What is a computable function? A functional dependence $y = f(x)$ between two physical quantities x and y is computable if, given the values of the quantity x, we can algorithmically predict the value of the quantity y.

In practice, as we have mentioned, we do not know the exact value of x, we only know estimates r_k for this value. We therefore need to be able, based on the estimate r_k for x, to compute the corresponding estimate s_ℓ for y. We also need to know how accurately we need to measure x to be able to estimate y with a given accuracy. Thus, we arrive at the following definition.

Definition 2. *A function $f : X \to Y$ is called* computable *if there exist two algorithms:*

- *the first algorithm, given a rational number $r \in X$ and a natural number ℓ, computes a $2^{-\ell}$-approximation to $f(r)$;*
- *the second algorithm, given a natural number ℓ, generates a natural number k such that $d(x, x') \le 2^{-k}$ implies*
$$d(f(x), d(x')) \le 2^{-\ell}.$$

Observable sets. We want to describe sets S – or, in other words, constraints – which are *observable* in the sense that, based on the observations (measurements), we can, in principle, determine whether a give state satisfies this constraint or not. Let us show that from the mathematical viewpoint, such sets have an interesting property – they are *closed* in the usual mathematical sense that:

- if a sequence of states $s_k \in S$ tends to a limit state s,
- then this limit state s should also belong to the set S.

From the mathematical viewpoint, the fact that the state s is equal to the limit of the states s_k means that for every $\varepsilon > 0$ there exists an integer K such that for all $k \ge K$, we have $d(s_k, s) \le \varepsilon$. In physical terms, this means that no matter how accurate our measurements, we will never be able to distinguish the state s from the appropriate state s_k. In other words, no matter how many measurements we perform, we cannot distinguish the state s from one of the physically possible states s_k. Since the state s is consistent with all possible measurements, this means that we should classify the state s as physically possible. Thus, the set S of physically possible states is indeed closed.

What is a computable set? Finally, let us describe what it means for a closed set S to be computable. We are interested in sets of states. From the mathematical viewpoint, we can have infinitely many possible states, characterized by all possible real-valued tuples $x = (x_1, \ldots, x_n)$. However, in practice, we only find an *estimate* of the actual state – by using measurements with a given accuracy.

Once we know the accuracy, then, in effect, we only have finitely many distinguishable states. For example, if the value of a quantity is known to be between 0 and 10, and we measure it with accuracy 0.1, then we cannot distinguish between the values like 1.02 and 1.07 the difference between them is below the instrument accuracy.

In this approximation, when we only have finitely many distinguishable states, a general set of states is represented simply a finite subset of this set of sets distinguishable within this accuracy. For each accuracy 2^{-k}, this (approximate) finite set S_k approximates the actual (infinite) set S in the following natural sense:

- each element $s \in S_k$ is 2^{-k}-close to some state $s' \in S$;
- vice versa, each element $s' \in S$ is 2^{-k}-close to some state $s \in S_k$.

This relation between the two sets can be described in terms of the *Hausdorff distance*

$$d_H(A, B) \overset{\text{def}}{=} \max\left(\max_{a \in A} d(a, B), \max_{b \in B} d(b, A)\right),$$

where $d(a, B) \overset{\text{def}}{=} \min_{b \in B} d(a, b)$, as $d_H(S_k, S) \le 2^{-k}$. Thus, we arrive at the following definition.

Definition 3. *A closed set $S \subseteq [L_1, U_1] \times \ldots \times [L_n, U_n]$ is called* computable *if there exists an algorithm that, given a natural number k, produces a finite list S_k of computable points which is 2^{-k}-close to S, i.e., for which*

$$d_H(S_k, S) \le 2^{-k}.$$

Comment. We consider closed bounded sets. It is known that closed bounded subsets of a finite-dimensional space $\mathbb{R}^n$ are *compact*. Because of this, computable sets are also known as *computable compact sets*; see, e.g., [1].

Known results about computable numbers, functions, and sets [1], [10], [13], [16]:

- For every two computable numbers $\ell < u$, we can, given a computable number x, check whether $x > \ell$ or $x < u$: to check this, it is sufficient to compute x, ℓ, and u with a sufficient accuracy.
- No algorithm is possible that, given a computable real number a, would check whether $a = 0$ or $a \neq 0$.
- No algorithm is possible that, given a computable real number a, would check whether $a \ge 0$ or $a < 0$.
- No algorithm is possible that, given a computable real number a, would check whether $a \le 0$ or $a > 0$.
- There is an algorithm that, given two computable tuples x and y, computes the distances

$$d_\infty(x, y) \overset{\text{def}}{=} \max(|x_1 - y_1|, \ldots, |x_n - y_n|) \text{ and}$$

$$d_2(x, y) \overset{\text{def}}{=} \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}.$$

- Minimum and maximum are computable.
- Composition of computable functions is computable. In particular, maximum and minimum of finitely many computable functions is computable.
- It is algorithmically possible, given a computable set S and a computable function $F(x)$, to compute $\max_{x \in S} F(x)$ and $\min_{x \in S} F(x)$.
- For every computable function $F(x, y)$ and for every computable set $S \subseteq X$, the functions $\max_{x \in S} f(x, y)$ and $\min_{x \in S} f(x, y)$ are computable functions of y.
- There exists an algorithm that, given a computable tuple x and a computable set S, returns the distance $d(x, S)$.
- For every two computable sets A and B, their union $A \cup B$ is also computable: namely, one can easily check that if A_k approximate A and B_k approximate B, then $A_k \cup B_k$ approximates $A \cup B$.
- For every two computable sets A and B, the corresponding set of pairs $A \times B$ is also computable: namely, one can easily check that if A_k approximate A and B_k approximate B, then $A_k \times B_k$ approximates $A \times B$ (in the sense of d_∞-metric).
- There exists an algorithm that, given a computable set S, a computable function f, and computable real numbers $a < b$, returns a computable number $\eta \in (a, b)$ for which the set $\{x \in S : f(x) \le \eta\}$ is computable.

III. Under Hard (Crisp) Constraints, All Four Fundamental Problems of Constraint Processing Are Not Algorithmically Solvable

Before we start describing our positive algorithmic results, let us first list the promised "negative results" – that if we treat constraints exactly, then for each of the four fundamental problems of constraint processing, no algorithm is possible which would solve all particular cases of the corresponding problem.

Proposition 1. *No algorithm is possible that, given a computable function $f(x_1, \ldots, x_n)$ and computable numbers $\underline{y}$ and $\overline{y}$, returns the set $\{x : \underline{y} \le f(x) \le \overline{y}\}$.*

Proof. The proof is by contradiction. One can easily check that the function $f(x_1) = \max(\min(x_1, 0), x_1 - 1)$ is computable. This function is equal:

- to x_1 for $x \le 0$,
- to 0 for $0 \le x \le 1$, and
- to $x - 1$ for $x \ge 1$.

For $\underline{y} = -1$ and $\overline{y} = a$, the set $\{x : \underline{y} \le f(x) \le \overline{y}\}$ is equal:

- to $[-1, 1 + a]$ when $a \ge 0$ and
- to $[-1, a]$ when $a < 0$.

Thus, the maximum M of the function $F(x_1) = x_1$ on this set is equal:

- to $1 + a$ for $a \ge 0$ and
- to a for $a < 0$.

In particular, for $|a| < 0.1$, we get:

- $M \ge 0.9$ when $a \ge 0$ and
- $M < 0.1$ when $a = 0$.

So, if we could algorithmically produce the set

$$\{x : \underline{y} \le f(x) \le \overline{y}\},$$

we would be able to estimate the value M and thus, to check whether a computable number is negative or non-negative – which is known to be impossible. The proposition is proven.

Proposition 2. *No algorithm is possible that, given two computable sets S_1 and S_2, computes their intersection.*

Proof. Let us take $S_1 = \{0, 1\}$. For every computable number a, let us take $S_2 = \{a, 1\}$. The intersection $S_1 \cap S_2$ is equal:

- to $\{0, 1\}$ when $a = 0$ and
- to $\{1\}$ when $a \neq 0$.

Thus, the minimum m of the function $F(x_1) = x_1$ over the intersection is equal:

- to 0 when $a = 0$ and
- to 1 when $a \neq 0$.

If the intersection was computable, then m would be computable too; by computing m with accuracy 0.1, we would be able to check whether $m = 0$ or $m = 1$ – and thus, check whether $a = 0$ or $a \neq 0$, and we know that this is not possible.

Proposition 3. *No algorithm is possible that, given computable sets $S \subseteq X$ and $R \subseteq X \times Y$, returns the composition $Y = R \circ S$.*

Proof. Let us take

$$R = \{(x, y) : (-1 \le x \le 0 \,\&\, y = 0) \vee$$

$$(0 \le x \le 1 \,\&\, -1 \le y \le 1)\}.$$

This set is clearly computable. For every computable number a, we can form a computable set $S = \{a\}$. Here:

- $R \circ S = \{0\}$ when $a < 0$ and
- $R \circ S = [-1, 1]$ when $a \ge 0$.

Thus, the maximum M of the function $F(y) = y$ over the set $R \circ S$ is equal:

- to 0 when $a < 0$ and
- to 1 when $a \ge 0$.

If we could compute the composition, we would be able to compute M and thus, decide whether $a < 0$ or $a \ge 0$ – and we know that this is impossible. The proposition is proven.

Proposition 4. *No algorithm is possible that, given computable sets S, A, and D, returns the set $\{a \in A : f(x, a) \in D$ for all $x \in S\}$ of actions that lead to the desired goal.*

Proof. Let us take $A = \{0, 1\}$, let $f(x, a) = a - x$, and let $D = [0, 1]$. In this case, we want to return the set R of all the actions $a \in \{0, 1\}$ for which $a \ge x$ for all $x \in S$. For each computable number $v \in (-1, 1)$, we can take $S = \{v\}$.

- When $v \le 0$, then $R = \{0, 1\}$.
- When $v > 0$, then $R = \{1\}$.

Thus, the minimum m of the function $F(a) = a$ over the set R is equal:

- to 0 if $v \le 0$ and
- to 1 if $v > 0$.

So, if we could compute the set R, we would be able to tell whether $v \le 0$ or $v > 0$ – and this is known to be impossible. The proposition is proven.

IV. Main Result: Under Soft Constraints, All Four Fundamental Problems of Constraint Processing Are Algorithmically Solvable

Constraints are actually soft. The above negative results assume that all constraints are *hard* (*crisp*), i.e., all the thresholds are exactly known. In reality, the thresholds like $\underline{y}$ and $\overline{y}$ are only approximately known: e.g., a bound on the measurement error can be 0.1, or it can be 0.101, from the physical viewpoint it is the same situation. So, if we find a solution for values which are slightly different from the original values $\underline{y}$ and $\overline{y}$, then this still solves the original physical problems.

Let us prove that due to this *softness* of constraints, all four constraint processing problems are algorithmically solvable.

Proposition 5. *There is an algorithm that, given a computable function $f(x_1, \ldots, x_n)$ and computable numbers $\underline{y} < \overline{y}$, and $\varepsilon > 0$, returns:*

- *a computable value $\underline{Y}$ which is ε-close to $\underline{y}$: $|\underline{Y} - \underline{y}| \le \varepsilon$;*
- *a computable value $\overline{Y}$ which is ε-close to $\overline{y}$: $|\overline{Y} - \overline{y}| \le \varepsilon$; and*
- *a computable set $\{x : \underline{Y} \le f(x) \le \overline{Y}\}$.*

Proof. One can easily check that the double inequality

$$\underline{y} \le f(x) \le \overline{y}$$

is equivalent:

- to $f(x) - \underline{y} \ge 0$ and $\overline{y} - f(x) \ge 0$, and thus,
- to $F(x) \le 0$, where $F(x) \stackrel{\text{def}}{=} \min(\overline{y} - f(x), f(x) - \underline{y})$, and
- to $-F(x) \le 0$.

According to one of the known results about computable sets which are listed above, for every $\varepsilon > 0$, there exists a $\eta \in (0, \varepsilon)$ for which the set $\{x : -F(x) \le \eta\}$ is computable.

The inequality $-F(x) \le \eta$ is equivalent to $F(x) \ge -\eta$. The smallest $F(x)$ of the two numbers $\overline{y} - f(x)$ and $f(x) - \underline{y}$ is greater than or equal to $-\eta$ if and only if both these numbers are $\ge -\eta$, i.e., if and only if $\overline{y} - f(x) \ge -\eta$ and

$$f(x) - \underline{y} \ge -\eta.$$

Here:

- the first inequality is equivalent to $f(x) \le \overline{Y}$, where $\overline{Y} \stackrel{\text{def}}{=} \overline{y} + \eta$ is ε-close to $\overline{y}$;
- the second inequality is equivalent to $\underline{Y} \le f(x)$, where $\underline{Y} \stackrel{\text{def}}{=} \underline{y} - \eta$ is ε-close to $\underline{y}$.

Thus, the inequality $-F(x) \le \eta$ is equivalent to

$$\underline{Y} \le f(x) \le \overline{Y}.$$

Hence, the set

$$\{x : \underline{Y} \le f(x) \le \overline{Y}\} = \{x : -F(x) \le \eta\}$$

is indeed computable. The proposition is proven.

Second problem: discussion and results. In the above first constraint processing problem, we had numbers (thresholds), so we described the softness of the corresponding constraint by allowing to slightly modify these numbers.

In the second constraint processing problem, we do not have thresholds, we only have sets, so we need to be able to modify sets.

Such a modification is possible if we take into account that each closed set S can be described as $\{x : d(x,S) = 0\}$, i.e., equivalently, as $\{x : d(x,S) \le t\}$, where $t = 0$. This description allows us to generate an approximate set by slightly modifying the corresponding threshold t.

Definition 4. *For each set S and for each real number $\eta > 0$, by an η-*neighborhood *$N_\eta(S)$, we means the set*

$$\{x : d(x,S) \le \eta\}.$$

Comment. One can easily check that the η-neighborhood $N_\eta(S)$ of the set S is η-close to this set:

$$d_H(S, d_\eta(S)) \le \eta.$$

Proposition 6. *There exists an algorithm that, given m computable sets $S_1, \ldots, S_m$, and a computable real number $\varepsilon > 0$, returns a computable number $\eta \in (0, \varepsilon)$ for which the intersection $N_\eta(S_1) \cap \ldots \cap N_\eta(S_m)$ is computable.*

Proof. A tuple x belongs to the intersection

$$N_\eta(S_1) \cap \ldots \cap N_\eta(S_m)$$

if and only if it belongs to all m η-neighborhoods $N_\eta(S_i)$, i.e., if and only if $d(x, S_i) \le \eta$ for all $i = 1, \ldots, m$.

A sequence of m numbers $d(x, S_i)$ is smaller than or equal to η if and only if the largest of them is smaller than or equal to η. Thus, the intersection can be described as $\{x : F(x) \le \eta\}$, where $F(x) \stackrel{\text{def}}{=} \max(d(x,S_1), \ldots, d(x,S_m))$.

The maximum $F(x)$ of m computable functions $d(x, S_i)$ is computable. Thus, according to the above property of computable sets, there exists an $\eta \in (0, \varepsilon)$ for which the set $\{x : F(x) \le \eta\}$ is computable – and, as we have shown, this set is exactly the desired intersection. The proposition is proven.

Proposition 7. *There exists an algorithm that, given computable sets $S \subseteq X$ and $R \subseteq X \times Y$ and a computable real number $\varepsilon > 0$, returns a computable number $\eta \in (0, \varepsilon)$ for which the composition $N_\eta(R) \circ N_\eta(S)$ is computable.*

Proof. The condition that $x \in S$ and $(x, y) \in R$ can be equivalently described as $d(x, S) = 0$ and $d((x, y), R) = 0$, and thus, as $\max(d(x,S), d((x,y),R)) = 0$.

So, the existence of such $x \in S$ is equivalent to $F(x) \le t \stackrel{\text{def}}{=} 0$, where $F(y) \stackrel{\text{def}}{=} \min\limits_{x \in S} \max(d(x,S), d((x,y),R))$.

The function $\max(d(x,S), d((x,y),R))$ is computable; thus, $F(y)$ is also computable, hence there exists an $\eta \in (0, \varepsilon)$ for which the set $\{y : F(y) \le \eta\}$ is computable.

The inequality $F(y) \le \eta$, i.e.,

$$\min_{x \in S} \max(d(x,S), d((x,y),R)),$$

is equivalent to the existence of x for which $\max(d(x,S), d((x,y),R)) \le \eta$, i.e., for which $d(x,S) \le \eta$ and $d((x,y),R) \le \eta$.

By the definition of a set neighborhood $N_\eta(A)$:

- the first inequality $d(x,S) \le \eta$ is equivalent to
$$x \in N_\eta(S), \text{ and}$$
- the second inequality $d((x,y),R) \le \eta$ is equivalent to $(x,y) \in N_\eta(R)$.

Thus, the condition $F(y) \le \eta$ is equivalent to the existence of $x \in N_\eta(S)$ for which $(x,y) \in N_\eta(R)$, i.e., to

$$y \in N_\eta(R) \circ N_\eta(S).$$

So, the computable set $\{y : F(y) \le \eta\}$ is equal to the composition $N_\eta(R) \circ N_\eta(S)$ – hence, this composition is indeed computable. The proposition is proven.

Proposition 8. *There exists an algorithm that, given computable $S \subseteq X$, A, and D, a computable function $f(x,a)$, and a computable real number $\varepsilon > 0$, returns a computable number $\eta \in (0, \varepsilon)$ for which the set*

$$\{a \in A : f(x,a) \in N_\eta(D) \textit{ for all } x \in S\}$$

is computable.

Proof. The condition that $f(x,a) \in D$ is equivalent to $d(f(x,a), D) \le t = 0$. Thus, the requirement that this inclusion holds for all $x \in S$ is equivalent to $F(a) \le t$, where

$$F(a) \stackrel{\text{def}}{=} \max_{x \in S} d(f(x,a), D).$$

The function $F(a)$ is computable; thus there exists a computable value $\eta \in (0, \varepsilon)$ for which the set $\{a \in A : F(a) \le \eta\}$ is computable.

The condition $F(a) = \max\limits_{x \in S} d(f(x,a), D) \le \eta$ is equivalent to the condition that $d(f(x,a), D) \le \eta$ for all $x \in S$, i.e., to the condition that for all $x \in S$, we have $f(x,a) \in N_\eta(D)$. The proposition is proven.

V. What If Some Measurements Are Faulty

Formulation of the problem. In the above analysis, we assumed that all the measurements are reliable. In this case, if we denote by S_i the set of all the states which are consistent with the i-th measurements, we can conclude that the actual state belong to the intersection $S_1 \cap \ldots \cap S_m$ of all these sets.

A measuring instrument is rarely 100% reliable. Sometimes, it mis-performs, resulting in a numerical value which is far away from the actual value of the corresponding physical quantity.

For example, when we measure a distance from an underwater autonomous robot to a beacon, we get a wrong result when instead of the sonar signal coming directly from the beacon, we observe the signal which was first reflected against some external surface; see, e.g., [6].

Usually, we know the reliability of the measuring instrument, i.e., we know what fraction of measurement results is unreliable. If this fraction is 10%, then we know that at least 90% of the measurements are correct. In general, based on the total number m of measurements and the fraction of faulty ones, we can estimate the number q of correct measurements. This way, we know that out of m measurements, at least q are correct. Thus, for some subset $I \subseteq \{1, \ldots, m\}$ of size $|I| = q$, the actual state s belongs to the intersection $\bigcap\limits_{i \in I} S_i$. The overall set S of possible states is thus equal to

$$S = \bigcup_{I : |I| = q} \left(\bigcap_{i \in I} S_i \right).$$

This set is called *q-relaxed intersection* [6].

The corresponding set is still computable – if we take into account that constraints are soft. Under hard constraints, the corresponding set S is, in general, not computable – indeed, we have shown that it is not computable even when $q = m$.

Under soft constraints, the desired set S is computable:

Proposition 9. *There exists an algorithm that, given m computable sets $S_1, \ldots, S_m$, an integer $q \le m$, and a computable real number $\varepsilon > 0$, returns a computable number $\eta \in (0, \varepsilon)$ for which the set $\bigcup\limits_{I : |I| = q} \left(\bigcap\limits_{i \in I} N_\eta(S_i) \right)$ is computable.*

Proof. A tuple x belongs to each intersection $\bigcap\limits_{i \in I} N_\eta(S_i)$ if it belongs to all q η-neighborhoods $N_\eta(S_i)$, i.e., if and only if $d(x, S_i) \le \eta$ for all $i \in I$.

A sequence of q numbers $d(x, S_i)$ is smaller than or equal to η if and only if the largest of them is smaller than or equal to η. Thus, x belongs to the intersection if and only if $F_I(x) \le \eta$, where $F_I(x) \stackrel{\text{def}}{=} \max\limits_{i \in I} d(x, S_i)$.

A tuple x belongs to the set $\bigcup\limits_{I : |I| = q} \left(\bigcap\limits_{i \in I} N_\eta(S_i) \right)$ if it belongs to one of the intersections, i.e., equivalently, if one of the value $F_I(x)$ is smaller than or equal to η. One of the values $F_I(x)$ is smaller than or equal to η if and only if the smallest of these numbers does not exceed η, i.e., if and only if $F(x) \le \eta$, where $F(x) = \max\limits_{I : |I| = q} F_I(x)$.

Each minimum $F_I(x)$ of q computable functions $d(x, S_i)$ is computable. Therefore, the maximum $F(x)$ of finitely many computable functions is also computable. Thus, according to the above property of computable sets, there exists an $\eta \in (0, \varepsilon)$ for which the set $\{x : F(x) \le \eta\}$ is computable – and, as we have shown, this set is exactly the desired union of intersections. The proposition is proven.

Discussion. While the desired set is computable, the above algorithm requires us to consider as many functions are there are subsets i of size q – and this number, for $q = k \cdot m$, exponentially grows with m. So, while the algorithm is *possible*, the above algorithm is clearly not *feasible*, since for even medium-size $m \approx 300$, the corresponding number 2^m of computational steps exceeds the lifetime of the Universe.

Is this because our algorithm is not perfect, or is this because the problem itself is complex? Our answer – as described by the following proposition – is that the itself problem is really complex, even for linear constraints. Specifically, we prove that this problem is *NP-hard* – i.e., that it is harder than all the problems from the reasonable class NP, in the sense that every problem form the class NP can be reduced to our problem; for exact definitions, see, e.g., [4], [9], [12]. Most computer scientists believe that P≠NP, and thus, that it is not possible to have a feasible algorithm for solving an NP-hard problem.

Proposition 10. *The following problem is NP-hard:*

- *given a set of m linear constraints and an integer q,*
- *produce the set of all the tuples which satisfy at least q out of m constraints.*

Proof. The following *subset sum* problem is known to be NP-hard: given $n + 1$ positive integers $s_1, \ldots, s_n$, and S, check whether S can be represented as a sum of some of the values s_i. Equivalently, we need to check whether it is possible to find values $x_i \in \{0, 1\}$ such that $\sum\limits_{i=1}^{n} s_i \cdot x_i = S$.

To prove that our problem is NP-hard, let us reduce the subset sum problem to our problem. Since the subset problem is NP-hard, this means that every problem from the class NP can be reduced to the subset sum problem; thus, if we can reduce the subset sum problem to our problem, it will follow that all problems from the class NP can be reduced to our problem – and thus, that our problem is indeed NP-hard.

Let us describe the desired reduction. For each set of values $s_1, \ldots, s_n, S$, let us form the following linear constraints:

$$x_1 = 0; x_1 = 1; \ldots x_n = 0; x_n = 1;$$

$$\sum_{i=1}^{n} s_i \cdot x_i = S \text{ (repeated } n \text{ times)}; \quad y = 0.$$

We require that out of these $m = 3n + 1$ constraints, at least $q = 2n$ are satisfied. We are interested in finding the set of all possible values y under this requirement.

The above $3n + 1$ constraints consist of three groups:

- the first $2n$ constraints are of the form $x_i = 1$ or $x_i = 1$;
- then, we have the same sum constraint repeated n times;
- and finally, we have an additional constraint $y = 0$.

Since each value x_i can be either 0 or 1 but not both, for each i, at most one of the constraints $x_i = 1$ and $x_i = 0$ can be satisfied. Thus, out of the first $2n$ constraints, at most n can be satisfied, and if exactly n are satisfied, then each value x_i is equal to either 0 or 1.

Since at most n constraints from the first group can be satisfied, the sum constraint has to be satisfied – otherwise, we will get at most $n + 1 < 2n$ constraints. So:

- If the subset sum problem has a solution, that we can get $2n$ constraints by selecting appropriate values x_i. In this case, the variable y can attain any value.

- On the other hand, if the subset sum problem does not have a solution, this means that we cannot have $2n$ constraints satisfied by simply picking appropriate values $x_i \in \{0,1\}$. Thus, to satisfy at least $q = 2n$ constraints, we must invoke the constraint $y = 0$.

Hence:

- if the given instance of the subset sum problem has a solution, then y can take any value;
- otherwise, if the given instance of the subset sum problem does not have a solution, then y can only take value 0.

So, if we know the range of possible values of y, we can check whether the given instance of the subset problem has a solution.

Thus, solving this particular case of our problem is equivalent to solving the given instance of the subset sum problem. This reduction proves that our problem is indeed NP-hard. The proposition is proven.

Application. A practical example of this approach is given in [6]: a problem of 2D-localization of a mobile underwater robot. To locate the robot, stationary sonars placed at known locations periodically send a ping signal in all directions; they send signals one after another, so that signals from different sonars do not get mixed up. When the sonar's signal reaches the robot, this signal gets reflected, and part of the reflected signal gets back to the emitting sonar. The sonar then measures the signal's "travel time" as the difference between the emission time and the time when the sonar received the reflected signal. During this travel time, the signal traveled to the robot and back. So, the overall path of the signal is double the distance d_i from the robot to the corresponding sensor i. Once we know the speed of sound, we can multiply the measured time interval by this speed, divide by two, and get the distance d_i to the robot.

In practice, we need to detect the reflected signal against the ever-present noise. Because of the noise, we can only determine the moment when the reflected signal appeared with some accuracy – thus, we can only measure the distance d_i with some accuracy. The manufacturer's specification for the sonar provide us with the upper bound Δ on the corresponding measurement error (provided, of course, that we are observing the reflection from the robot and not from some other object). Thus, if the measured distance to the i-th sonar is $\widetilde{d}_i$, then the actual (unknown) distance d_i can take any value from the interval $[\widetilde{d}_i - \Delta, \widetilde{d}_i + \Delta]$.

For each sonar, we thus conclude that the robot is located in the ring S_i formed by the two circles centered around this sonar: the ring between the circle corresponding to distance $\widetilde{d}_i - \Delta$ and the circle corresponding to the distance $\widetilde{d}_i + \Delta$. If all the recorded values $\widetilde{d}_i$ corresponded to the robot, then we could find the set S of possible locations of the robot as the intersection of the sets S_i corresponding to all m sonars. In real life, some measurements do come from other objects; in this case, some of the sets S_i reflect locations of these other objects, and thus, the overall intersection may be empty. We therefore need to take into account that some of the measurements are faulty.

To locate the robot, we use a Guaranteed Outlier Minimal Number Estimator (GOMNE) described in [5], [7], [8]. This algorithm first finds the largest possible value q for which the intersection of q sets S_i is non-empty, then finds the corresponding q-relaxed intersection. To compute the corresponding intersections, GOMNE uses SIVIA (Set Inversion via Interval Analysis), an algorithm described in [5].

Simulations show that in more than 90% of the cases, the resulting algorithm finds the correct location of the robot, which is much more efficient than for the previously known methods of locating underwater robots.

Acknowledgment

This work was supported in part by the US National Science Foundation grants 0953339, HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721. The work was performed when Quentin Brefort was visiting the University of Texas at El Paso. This visit was supported by ENSTA-Bretagne.

References

[1] E. Bishop and D. S. Bridges, *Constructive Analysis*, Springer, New York, 1985.

[2] M. Ceberio and V. Kreinovich (eds.), *Constraint Programming and Decision Making*, Springer Verlag, Berlin, Heidelberg, 2014.

[3] R. Dechter, *Constraint Processing*, Morgan Kaufmann, San Francisco, California, 2003.

[4] M. E. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[5] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*, Springer Verlag, London, 2001.

[6] L. Jaulin, A. Stancu, and B. Desrochers, "Inner and outer approximations of probabilistic sets", *Proceedings of the American Society of Civil Engineers (ASCE) Second International Conference on Vulnerability and Risk Analysis and Management ICVRAM'2014 and Sixth International Symposium on Uncertainty Modelling and Analysis ISUMA'2014*, Liverpool, UK, July 13–16, 2014, to appear.

[7] L. Jaulin and E. Walter, "Guaranteed robust nonlinear minimax estimation", *IEEE Transaction on Automatic Control*, 2002, Vol. 47, No. 11, pp. 1857–1864.

[8] L. Jaulin, E. Walter and O. Didrit, "Guaranteed robust nonlinear parameter bounding", *Proceedings of Symposium on Modelling, Analysis and Simulation, part of IMACS Multiconference on IMACS Multiconference, Computational Engineering in Systems Applications CESA'96*, Lille, France, July 9–12, 1996, Vol. 2, pp. 1156–1161.

[9] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[10] B. A. Kushner, *Lectures on Constructive Mathematical Analysis*, Amer. Math. Soc., Providence, Rhode Island, 1984.

[11] C. Lecoutre, *Constraint Networks: Techniques and Algorithms*, ISTE, London, UK, and Wiley, Hoboken, New Jersey, 2009.

[12] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.

[13] M. Pour-El and J. Richards, *Computability in Analysis and Physics*, Springer-Verlag, New York, 1989.

[14] S. G. Rabinovich, *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, Berlin, 2005.

[15] E. Tsang and T. Fruehwirth, *Foundations of Constraint Satisfaction*, Books On Demand Publ., 2014.

[16] K. Weihrauch, *Computable Analysis*, Springer-Verlag, Berlin, 2000.