Constraint Problems: Computability Is Equivalent to Continuity

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

Abstract

In many practical situations, we would like to compute the set of all possible values that satisfy given constraints. It is known that even for computable (constructive) constraints, computing such set is not always algorithmically possible. One reason for this algorithmic impossibility is that sometimes, the dependence of the desired set on the parameters of the problem is not continuous, while all computable functions of real variables are continuous. In this paper, we show that this discontinuity is the only case when the desired set cannot be computed. Specifically, we provide an algorithm that computes such a set for all the cases when the dependence is continuous.

1 Constraint Satisfaction and Constraint Optimization: From a Practical Problems to Constructive Mathematics

Constraints are ubiquitous. To describe a state of a physical system, we measure the values of the physical quantities characterizing this system – its coordinates, its velocity, its temperature, etc. The state can be then characterized by the tuple $x = (x_1, \ldots, x_n)$ consisting of the results $x_1, \ldots, x_n$ of these measurements.

Not every tuple of n real numbers can represent a state, there are restrictions (constraints) on possible combinations $x = (x_1, \ldots, x_n)$. Some of these constraints are inequalities, i.e., have the type $f(x) \geq c$ or $f(x) \leq c$: e.g., the velocity of a system cannot exceed the speed of light; the entropy of the closed system cannot be smaller than the initial value of its entropy, etc. Other constraints are equalities, i.e., have the type $g(x) = v$: e.g., the energy of a closed system must be always equal to the initial value of this energy.

Simplification. In principle, we can have inequality of different types: of the type $f(x) \geq c$, of the type $f(x) \leq 0$, and double-sided inequalities $a \leq f(x) \leq b$. To simplify our analysis, we will reduce them to inequalities of the same type $f(x) \geq c$. This reduction is straightforward:

- a double inequality $a \leq f(x) \leq b$ can be represented as a pair of inequalities $a \leq f(x)$ and $f(x) \leq b$, and
- an inequality $f(x) \leq c$ can be equivalently reformulated as $-f(x) \geq -c$.

Comment. It should be mentioned that from the purely mathematical viewpoint, an equality $f(x) = c$ can also be described as a degenerate case of a double inequality $c \leq f(x) \leq c$. We will use this fact in our mathematical analysis (and in our proofs).

However, it turns out that this representation is not always useful when we analyze computability. As a result, in the following text, we will consider the general case when we can have *both* inequalities and equalities.

Practice-motivated constraints are computable. In all practical cases, the constraints are computable: given a state x, we can compute the corresponding values $f(x)$ and $g(x)$. Since this value may be $\sqrt{2}$ or π or any other number which cannot be exactly represented in modern computers, we can reword this statement in more precise terms: that given a state x and the desired computation accuracy $\varepsilon > 0$, we can compute the values $f(x)$ and $g(x)$ with the desired accuracy – for example, we can compute the value r for which $|r - f(x)| \leq \varepsilon$.

First problem: constraint satisfaction. Once we have found all the constraints that the actual state x must satisfy, i.e., constraints of the type $f_1(x) \geq c_1$, ..., $f_n(x) \geq c_n$, $g_1(x) = v_1$, ..., $g_m(x) = v_m$, it is desirable to describe all possible tuples x that satisfy these constraints.

What does it mean "to describe all possible tuples x"? Strictly speaking, the constraints themselves already provide the description of the set S of all possible tuples that satisfy these constraints. The problem with this description is that it is *implicit*; what we need is an *explicit* description of this set.

In some cases, this set consists of a single tuple. For example, we may have a constraint $f(x) = c$ which has exactly one solution. In such a case, when the set S consists of a single tuple x, it is very clear what is meant by explicitly describing this set — we need to produce this tuple x. In other words, given the desired computation accuracy ε, we must compute a tuple r which is ε-close to x, i.e., for which $d(r, x) \leq \varepsilon$ (e.g., in the sense of the usual Euclidean metric d).

Similarly, if the set S consists of a finite number of solutions, an explicit description means that we have to explicitly list all these solutions.

In many practical situations, however, the set S is infinite. This is usually true, e.g., when all the constraints are inequality constraints. For example, if the only constraint on the velocity v is that $|v| \leq c$, then the set S of possible

values of velocity that satisfy this constraint is the entire interval $[-c, c]$. At first glance, an interval contains infinitely many points, we cannot list them all. However, if we take into account that we only want to list the points with a given accuracy, this becomes feasible: for a given value $\varepsilon > 0$, we can provide a list of values $x^{(1)}, \ldots, x^{(N)} \in [-v, v]$ such that every point from this interval is ε-close to one of these points. For example, we can take values $x^{(k)} = \dfrac{k}{N}$ for a sufficiently large N.

In general, to describe the set S means that for any given accuracy $\varepsilon > 0$, we should be able to produce a finite list $L \subseteq$ such that for every element $s \in S$, there is a point $x \in L$ for which $d(x, s) \le \varepsilon$. Such a set L is called an *ε-net*; so, what we are interested in is producing an algorithm that, given $\varepsilon > 0$, would generate an ε-net for this set S.

Why constructive mathematics. We are interested in algorithms for generating mathematical objects. So, the first question that we ask is when such an algorithm is possible and when not. The reason why this question is important is that, as we will see, such an algorithm is not always possible.

In other words, the first question is whether the desired set S is computable – in some reasonable sense informally described above (formal definitions will be given in the next section). A general analysis of the existence of such algorithms (i.e., of computability of different mathematical objects) is known as *constructive mathematics.* Thus, to check solvability of constraint satisfaction problems, we need to use constructive mathematics.

Second problem: optimization under constraints. In practice, it is often desirable not only to describe the set S of possible states of a system (i.e., the set of all tuples x that satisfy a given constraint), it is also desirable to find the largest value of a certain quantity $h(x)$ depending on this state. For example, when we design a chemical reactor, we not only need to know the set of all its possible states x, we would also like to know the largest pressure $h(x)$ for all possible states – so as to build a reservoir that will be able to withstand this largest pressure.

In mathematical terms, this means that we need to find the largest possible value of a given function $h(x)$ under the given constraints $f_1(x) \ge c_1$, $\ldots$, $f_n(x) \ge c_n$, $g_1(x) = v_1$, $\ldots$, $g_m(x) = v_m$. To be more precise, when we say "to find", what we really mean is "to compute". From this viewpoint, the first natural question is when it is possible to compute this maximum. This computability question also falls within the scope of constructive mathematics.

2 Definitions of Computable (Constructive) Objects: Reminder

What we do in this section. To analyze the above two problems, we need to recall the definitions of computable (constructive) objects – and the basic

properties of these definitions. These definitions and properties will be listed in this section – as well as motivations for these definitions. For more details, see, e.g., [18, 20] (see also [1, 3, 4, 5, 6, 7, 11, 12]).

Readers who are familiar with the main ideas and results of constructive mathematics can skip this section and go directly to the next section, when we start formulating and proving our computability results.

Computable real numbers: motivations. In practice, many quantities such as weight, speed, etc., are characterized by real numbers. To get information about the corresponding value x, we perform measurements. Measurements are never absolute accurate. As a result of each measurement, we get a measurement result $\widetilde{x}$; for each measurement, we usually also know the upper bound Δ on the (absolute value of) the measurement error $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$: $|x - \widetilde{x}| \le \Delta$.

To fully characterize a value x, we must measure it with a higher and higher accuracy. As a result, when we perform measurements with accuracy 2^{-n} with $n = 0, 1, \ldots$, we get a sequence of rational numbers r_n for which $|x - r_n| \le 2^{-n}$.

From the algorithmic viewpoint, we can view this sequence as an oracle that, given an integer n, returns a rational number r_n.

Computable real numbers: the resulting definition. A real number x is called *computable* if there exists an algorithm that, given a natural number n, returns a rational number r_n which is 2^{-m}-close to x. Equivalently, instead of specifying the sequence 2^{-n}, we can require the existence of an algorithm that, given a rational number $\varepsilon > 0$, produces a rational number which is ε-close to x.

Computable functions of real variables. Similarly, we can define a function $f(x)$ from real numbers to real numbers as a mapping that, given an integer n, a rational number x_m and its accuracy m, produces either a message that this information is insufficient, or a rational number y_n which is 2^{-n}-close to all the values $f(x)$ for $d(x, x_m) \le 2^{-m}$ – and for which, for every x and for each desired accuracy n, there is an m for which a rational number y_n is produced. We can also define a computable function $f(x_1, \ldots, x_k)$ of several real variables.

Computable metric spaces: motivations. A metric space (X, d) is a set on which we have a metric d, i.e., a function which assigns, to every two points $x, x' \in X$, a real number $d(x, x')$ called a *distance* between x and x'. How can we describe points of a computable metric space? Let us recall that a computable real number is described by its (rational) approximations. Similarly, it is reasonable to describe points of an arbitrary set X by their approximations.

From the computational viewpoint, each approximation can be represented in the computer, and thus, is encoded by a finite sequence of 0s and 1s. There are countably many such sequences, so we can describe these approximations as a sequence $x_1, \ldots, x_n, \ldots$ of points of X. Every element of the set X can be approximated, with arbitrary accuracy, by such approximations. Thus, every

point from the metric space can be represented as a limit of some subsequence of $\{x_n\}$ – i.e., in topological terms, this subsequence must be *dense* in the original space X.

Thus, we arrive at the following definition.

Computable metric spaces: resulting definition. By a *a computable metric space*, we mean a triple $(X, d, \{x_n\})$, where (X, d) is a metric space, $\{x_1, x_2, \ldots, x_n, \ldots\}$ is a dense subset of X, and there exists an algorithm that, given two natural numbers i and j, computes the distance $d(x_i, x_j)$.

Discussion. In other words, we have an algorithm that, given i, j, and an accuracy k, computes the 2^{-k}-rational approximation to $d(x_i, x_j)$. Similar to the previous examples, when we say that a computable metric space is given, we mean that we are given an algorithm that computes $d(x_i, x_j)$.

In particular, the set of all real numbers with a standard metric $d(x, x') = |x - x'|$ and all rational numbers as approximations $\{x_n\}$ is a computable metric space.

Computable points in computable metric spaces. Similar to the definition of a computable real number, a computable point x in a metric space can be defined by the existence of an algorithm which returns the corresponding approximations to x.

To be more precise, a point $x \in X$ of a computable metric space $(X, d, \{x_n\})$ is called *computable* if there exists an algorithm that transforms an arbitrary natural number k into a natural number i for which $d(x, x_i) \le 2^{-k}$. It is said that this algorithm *computes* the point x.

When we say that a computable point is given, we mean that we are given an algorithm that computes this point.

It is easy to show that a distance between two computable points x and y is computable.

Computable functions: general case. Many real-life quantities x, y are related by an (efficiently computable) functional relation $y = f(x)$. For example, the volume V of a cube is equal to the cube of its linear size s: $V = f(s) = s^3$. This means that, once we know the linear size, we can compute the volume.

At every moment of time, we can only know an approximate value of the actual quality $x \in X$. Thus, to be able to compute $f(x)$ with a given accuracy 2^{-k}, we must:

- be able to tell with what accuracy we need to know x, and then
- be able to use the corresponding approximation to compute $f(x)$.

We thus arrive at the following definition.

Computable function: definition. A function $f : X \to X'$ from a computable metric space $(X, d, \{x_n\})$ to a computable metric space $(X', d', \{x'_n\})$ is called *computable* if there exist two algorithms U_f and φ with the following properties:

- the algorithm φ takes a natural number k and produces a natural number $\ell = \varphi(k)$ such that $d(x, y) \le 2^{-\ell}$ implies that $d'(f(x), f(y)) \le 2^{-k}$;
- U_f takes two natural numbers n and k and produces a 2^{-k}-approximation to $f(x_n)$, i.e., a point x'_ℓ for which $d'(x'_\ell, f(x_n)) \le 2^{-k}$.

Computable compact sets: motivations. Let us recall that a metric space X is compact if and only if it is complete and totally bounded; see, e.g., [8, 21]. Here, *complete* means that every converging sequence of points X, i.e., every sequence y_n for which $d(y_n, y_m) \le 2^{-n} + 2^{-m}$ has a limit point in X, and *totally bounded* means that for every $\varepsilon > 0$, there exists a finite ε-net, i.e., a finite set of points $\{z_1, \ldots, z_N\}$ such that every point $x \in X$ is ε-close to one of the points z_i. It can be proven that:

- to check compactness, it is sufficient to check the existence of ε-nets only for some sequence of values $\varepsilon_n \to 0$, in particular, for $\varepsilon_n = 2^{-n}$;
- it is always possible to select an ε-net from the dense subset $\{x_n\} \subseteq X$; and
- to check that a given finite set F is indeed an ε-net, it is sufficient to check that every point x_n from the dense set is ε-close to one of the points of F.

Because of these results, the constructive analogues of the notion of compactness are usually formulated as the possibility to constructively design an 2^{-k}-net for a given k.

Computable compact set: definition. A computable metric space $(X, d, \{x_n\})$ is called *a computable compact space* if there exists an algorithm that, given an arbitrary natural number k, returns a finite set of indices $F_k \subset \{1, 2, \ldots, n, \ldots\}$ such that for every i there is a $f \in F_k$ for which $d(x_i, x_f) \le 2^{-k}$.

Properties of computable compact sets. An important feature of computable compact spaces X is that for every computable function $f : X \to R$ from X to real numbers, it is possible to efficiently compute its maximum and its minimum.

Another property that we will use is that for every computable function $f(x)$ from a computable compact set X to real numbers, and for every two computable numbers $a < b$, there exists a computable real number $c \in (a, b)$ for which the level set $\{x : f(x) \ge c\}$ is a computable compact set; see, e.g., [4, 5].

3 Hausdorff Metric: Definition and Basic Properties

The *Hausdorff distance* $d_H(A, B)$ between the two sets A and B in a metric space is defined as the infimum of all the values $\varepsilon > 0$ for which the following two properties hold:

- for every point a from the set A there is a point b from the set B for which $d(a, b) \leq \varepsilon$; and
- for every point b from the set B there is a point a from the set A for which $d(a, b) \leq \varepsilon$.

This definition is equivalent to

$$d_H(A, B) = \max\left(\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\right).$$

For compact sets A and B, infimum can be replaced by minimum, and supremum by maximum, so

$$d_H(A, B) = \max\left(\max_{a \in A} \min_{b \in B} d(a, b), \min_{b \in B} \max_{a \in A} d(a, b)\right).$$

It is known (and easy to prove) that Hausdorff distance is a *metric*, i.e., that it is symmetric ($d_H(A, B) = d_H(B, A)$) and satisfies the triangle inequality $d_H(A, C) \leq d_H(A, B) + h_H(B, C)$.

Hausdorff distance has the following two properties:

Proposition 1. *If $A \subseteq B \subseteq C$, then $d_H(A, B) \leq d_H(A, C)$ and $d_H(B, C) \leq d_H(A, C)$.*

Proof. To prove that $d_H(A, B) \leq d_H(A, C)$, we need to prove:

- that for every point $a \in A$, there is a point $b \in B$ for which $d(a, b) \leq d_H(A, C)$, and
- that for every $b \in B$, there is a point $a \in A$ for which $d(a, b) \leq d_H(A, C)$.

Indeed:

- On the one hand, since $A \subseteq B$, for given $a \in A$, we can simply take $b = a$, then $d(a, b) = 0 \leq d_H(A, C)$.
- On the other hand, if we have a point $b \in B$, then, since $B \subseteq C$, we have $b \in C$. By definition of the Hausdorff distance $d_H(A, C)$, we conclude that there exists a point $a \in A$ for which $d(a, b) \leq d_H(A, C)$.

Thus, indeed, $d_H(A, B) \leq d_H(A, C)$.

The proof that $d_H(B, C) \leq d_H(A, C)$ is similar. The proposition is proven.

Proposition 2. *If $d_H(A, X_\alpha) \le \varepsilon$ for all α, then $d_H\left(A, \bigcup\limits_\alpha X_\alpha\right) \le \varepsilon$.*

Proof. If $a \in A$, then, for every α, since $d_H(A, X_\alpha) \le \varepsilon$, there exists a point $x \in X_\alpha$ for which $d(a, x) \le \varepsilon$. Since $x \in X_\alpha$, we also have $x \in \bigcup\limits_\alpha X_\alpha$, so there exists a point $x \in \bigcup\limits_\alpha X_\alpha$ for which $d(a, x) \le \varepsilon$.

On the other hand, let $x \in \bigcup\limits_\alpha X_\alpha$. This means that for some α, we have $x \in X_\alpha$. Since $d_H(A, X_\alpha) \le \varepsilon$, this means that there exists a point a for which $d(a, x) \le \varepsilon$. The proposition is proven.

From the definition of Hausdorff distance, once can easily see that a finite list $L \subseteq S$ is an ε-net for S if and only if $d_H(L, S) \le \varepsilon$. This reformulation of the notion of an ε-net enables us to compute Hausdorff distance between computable compacts:

Proposition 3. *There exists an algorithm that, given two computable compact sets A and B, computes the Hausdorff distance $d_H(A, B)$.*

Comment. In other words, Hausdorff distance is computable.

Proof of Proposition 3. For computable finite lists L and L', the above formula for the distance $d_H(L, L')$ provides an explicit way to compute this distance – since minimum and maximum of two computable numbers is also computable. Thus, to compute the distance between two computable computable compact sets A and B with a given rational accuracy $\varepsilon > 0$, we can do the following:

- first, we use the fact that A is a computable compact set and compute a $\frac{\varepsilon}{3}$-net L_A for A; here, $d_H(A, L_A) \le \frac{\varepsilon}{3}$;
- similarly, we use the fact that B is a computable compact set and compute a $\frac{\varepsilon}{3}$-net L_B for B; here, $d_H(B, L_B) \le \frac{\varepsilon}{3}$;
- finally, we compute the Hausdorff distance $d_H(L_A, L_B)$ between the computable finite lists L_A and L_B with the accuracy $\frac{\varepsilon}{3}$, i.e., we compute a rational number r for which $|r - d_H(L_A, L_B)| \le \frac{\varepsilon}{3}$.

Let us show that $|r - d_H(A, B)| \le \varepsilon$, i.e., that $r \le d_H(A, B) + \varepsilon$ and $d_H(A, B) \le r + \varepsilon$.

Indeed, on the one hand, by the choice of r, we have $r \le d_H(L_A, L_B) + \frac{\varepsilon}{3}$. Due to the triangle inequality, we have

$$d_H(L_A, L_B) \le d_H(L_A, A) + d_H(A, B) + d_H(B, L_B).$$

Since $d_H(A, L_A) \le \dfrac{\varepsilon}{3}$ and $d_H(B, L_B) \le \dfrac{\varepsilon}{3}$, we have $d_H(L_A, L_B) \le d_H(A, B) + \dfrac{2\varepsilon}{3}$. Thus, $r \le d_H(L_A, L_B) + \dfrac{\varepsilon}{3}$ implies that

$$r \le \left(d_H(A, B) + \frac{2\varepsilon}{3}\right) + \frac{\varepsilon}{3} = d_H(A, B) + \varepsilon.$$

On the other hand, we have $d_H(L_A, L_B) \le r + \dfrac{\varepsilon}{3}$. From the triangle inequality, we conclude that

$$d_H(A, B) \le d_H(A, L_A) + d_H(L_A, L_B) + d_H(L_B, B) \le d_H(L_A, L_B) + \frac{2\varepsilon}{3},$$

and hence, that $d_H(A, B) \le \left(r + \dfrac{\varepsilon}{3}\right) + \dfrac{2\varepsilon}{3} = r + \varepsilon$.

The proposition is proven.

Proposition 4. *Let us assume that A is a compact set in a computable metric space for which there is an algorithm that, given a natural number n, produces a computable compact A_n in such a way that $d_H(A_n, A) \le 2^{-n}$. Then, A is a computable compact A.*

Comment. In other words, if we have an algorithm that, given a rational number ε, computes an ε-approximation to the desired compact set A, then this set A is itself computable.

Proof of Proposition 4. To construct a computable compact set means that, given a rational number $\varepsilon > 0$, we should be able to construct a ε-net for the desired compact set A. Let k be such that $6 \cdot 2^{-k} \le \varepsilon$, and let L_k be a 2^{-k}-net for the constructive compact set A_k. We will construct a finite list that contains exactly as many points as the list L_k – one point for each point $a_k \in L_k$ – and we will show that the resulting list is the desired ε-net for the limit compact set A.

Let a_k be a point from the list $L_k \in A_k$.

- Since $d_H(A_k, A_{k+1}) \le 2^{-k} + 2^{-(k+1)} < 2 \cdot 2^{-k}$, we can construct a point $a_{k+1} \in A_{k+1}$ for which $d(a_k, a_{k+1}) \le 2 \cdot 2^{-k}$.
- Similarly, since $d_H(A_{k+1}, A_{k+2}) \le 2^{-(k+1)} + 2^{-(k+2)} < 2 \cdot 2^{-(k+1)}$, we can construct a point $a_{k+2} \in A_{k+2}$ for which $d(a_{k+1}, a_{k+2}) \le 2 \cdot 2^{-(k+1)}$, etc.

As a result, we have a sequence of points $a_k, a_{k+1}, \ldots, a_\ell, \ldots$ for which $d(a_\ell, d_{\ell+1}) \le 2 \cdot 2^{-\ell}$. Thus, for every $\ell < \ell'$, we get

$$d(a_\ell, a_{\ell'}) \le d(a_\ell, a_{ell+1}) + d(a_{\ell+1}, a_{\ell+2}) + \ldots + d(a_{\ell'-1}, a_{\ell'}) \le$$

$$2 \cdot (2^{-\ell} + 2^{-\ell-1} + \ldots + 2^{-(\ell'-1)}).$$

Here,

$$2^{-\ell} + 2^{-\ell-1} + \ldots + 2^{-(\ell'-1)} \le 2^{-\ell} + 2^{-\ell-1} + \ldots = 2 \cdot 2^{-\ell};$$

thus, $d(a_\ell, a_{\ell'}) \le 4 \cdot 2^{-\ell}$. The sequence a_ℓ is therefore a Cauchy sequence, and therefore, it has a limit a_∞. This limit point belongs to the limit compact A. By taking $\ell' \to \infty$, we conclude that $d(a_\ell, a_\infty) \le 4 \cdot 2^{-\ell}$.

Thus, for every rational number δ, we can compute a δ-approximation to a_∞ as follows: find a natural number ℓ for which $4 \cdot 2^{-\ell} \le \delta$, and take a_ℓ as the desired approximation. So, a_∞ is a computable point.

To complete our proof, we need to show that the resulting list of points $a_\infty, b_\infty, \ldots$ indeed forms an ε-net for the limit compact set A, i.e., that for each point $a \in A$, there is the corresponding limit point which is ε-close to this point a. Indeed, from $d_H(A_k, A) \le 2^{-k}$, by definition of the Hausdorff metric, we conclude that there exists a point $a' \in A_k$ for which $d(a, a') \le 2^{-k}$. By definition of a 2^{-k}-net, there exists a point $a_k \in L_k$ for which $d(a', a_k) \le 2^{-k}$. For the corresponding limit point a_∞, we get $d(a_k, a_\infty) \le 4 \cdot 2^{-k}$. Thus, by the triangle inequality, we get

$$d(a, a_\infty) \le d(a, a') + d(a', a_k) + d(a_k, a_\infty) \le 2^{-k} + 2^{-k} + 4 \cdot 2^{-k} = 6 \cdot 2^{-k} \le \varepsilon.$$

The proposition is proven.

4 In General, the Set of Solutions Is Not Computable

Reminder. To compute a solution set means that for every accuracy $\varepsilon > 0$, we can compute an ε-approximation (i.e., an ε-net) to the desired solution set. From this viewpoint, our first algorithmic impossibility result takes the following form:

Proposition 5. *There exists a computable function $f(x)$ from an interval $[a, b]$ to real numbers for which no algorithm is possible that, given computable real numbers c and ε, returns an ε-net for the set $\{x : f(x) \ge c\}$.*

Discussion. In other words, it is not possible to have a computable mapping that maps a computable real number c into a computable compact set

$$\{x : f(x) \ge c\}.$$

The proof of this statement comes from the following lemma:

Lemma 1. *There exists a computable function $f(x)$ for which the mapping $c \to \{x : f(x) \ge c\}$ is not continuous (in Hausdorff metric).*

Proof of Lemma 1. Indeed, we can take the following piecewise-linear function on the interval $[0, 4]$:

- $f(x) = 1 - x$ for $0 \le x \le 1$,
- $f(x) = x - 1$ for $1 \le x \le 2$, and
- $f(x) = 3 - x$ for $2 \le x \le 4$.

For this function, for $c < 0$, $\{x : f(x) \ge c\} = [3 + c, 4]$, while for $c = 0$, we have $\{x : f(x) \ge c\} = \{1\} \cup [3, 4]$. Thus, when $c < 0$, we have

$$d_H(\{x : f(x) \ge c\}, \{x : f(x) \ge 0\}) = 2 + c.$$

When $c \to 0$, this distance tends to 2, and not to the limit distance

$$d_H(\{x : f(x) \ge 0\}, \{x : f(x) \ge 0\}) = 0.$$

The lemma is proven.

Proof of Proposition 5. Proposition 5 now follows from the known fact that every computable function is continuous; see, e.g., [12, 20].

Comment 1. For a single function $f(x)$, continuity of the level sets $\{x : f(x) \ge c\}$ and $\{x : f(x) = c\}$ means that the function $f(x)$ has no local maxima or minima; a general topological analysis of such level sets is performed in *Mores theory*; see, e.g., [13, 14, 17].

Comment 2. In the above example, the mapping $c \to \{x : f(x) \ge c\}$ are not computable if we allow arbitrary computable values c. If we only allow *algebraic* values c (i.e., values which satisfy a polynomial equation $P(c) = 0$ with non-zero integer coefficients) and *algebraic* functions $y = f(x)$ (i.e., functions which satisfy a polynomial equation $P(x, y) = 0$ with non-zero integer coefficients), then we can use the Tarski-Seidenberg algorithm to explicitly describe the level set as a computable compact set; see, e.g., [2, 15, 19].

5 Continuity Implies Computability

Discussion. In the above example, discontinuity prevented computability of the solution set. A natural question is: what if the dependence *is* continuous, will we then be able to compute the solution?

It turns out that this is indeed the case.

Case when we only have inequalities. The situation is the simplest in the case when we only have inequalities. In this case, as the following result shows, it is sufficient to know that the dependence is continuous:

Proposition 6. *Let $f_1, \ldots, f_n$ be computable functions from a computable compact set X to real numbers. Let $[\underline{c}_1, \overline{c}_1]$, $\ldots$, $[\underline{c}_n, \overline{c}_n]$ be computable intervals for which the set $\{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n\}$ continuously depends on $c_1, \ldots, c_n$. Then, there exists an algorithm that:*

- *given computable values $c_1 \in (\underline{c}_1, \overline{c}_1)$, $\ldots$, $c_n \in (\underline{c}_n, \overline{c}_n)$,*
- *computes the set $\{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n\}$.*

Proof. We want to prove that the desired level set

$$A \stackrel{\text{def}}{=} \{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n\}$$

is a computable compact set. According to Proposition 4, to prove this, it is sufficient to provide an algorithm that, given a rational number $\varepsilon > 0$, produces a computable compact set A' for which $d_H(A', A) \leq \varepsilon$. In other words, it is sufficient to be able to compute, for each ε, an ε-approximation to the desired level set A.

Indeed, each inequality $f_i(x) \geq c_i$ is equivalent to the condition that difference between the left-hand and the right-hand sides of this inequality is non-negative: $f_i(x) - c_i \geq 0$. Thus, all n inequalities are satisfied if and only if the smallest of these differences is non-negative, i.e., when $f(x) \geq 0$, where we denoted $f(x) \stackrel{\text{def}}{=} \min(f_1(x) - c_1, \ldots, f_n(x) - c_n)$. So, $A = \{x : f(x) \geq 0\}$. Since the functions $f_i(x)$ and the values c_i are computable, the function $f(x)$ is computable as well.

By the known property of a computable compact, for every integer k, we can algorithmically find values $-\delta_- \in (-2^{-k}, 0)$ and $\delta_+ \in (0, 2^{-k})$ for which the level sets $S_k^- \stackrel{\text{def}}{=} \{x : f(x) \geq -\delta_-\}$ and $S_k^+ \stackrel{\text{def}}{=} \{x : f(x) \geq \delta_+\}$ are computable compacts. Since the sets S_k^- and S_k^+ are computable compacts, we can compute the Hausdorff distance $d_k \stackrel{\text{def}}{=} d_H(S_k^-, S_k^+)$.

In terms of the original functions $f_i(x)$, these computable level sets take the form

$$S_k^- = \{x : f_1(x) \geq c_1 - \delta_-, \ldots, f_n(x) \geq c_n - \delta_-\}$$

and

$$S_k^+ = \{x : f_1(x) \geq c_1 + \delta_+, \ldots, f_n(x) \geq c_n + \delta_+\}.$$

Since the dependence of the original level sets on c_i is continuous, the Hausdorff distance d_k between these two level sets S_k^- and S_k^+ tends to 0 as $k \to \infty$. Thus, if we repeat this procedure for $k = 1, 2, \ldots$, we will eventually find a value k for which $d_k = d_H(S_k^-, S_k^+) \leq \dfrac{\varepsilon}{2}$.

Since $S_k^+ \subseteq A \subseteq S_k^+$, Proposition 1 now implies that $d_H(A', A) \leq 2^{-k}$ for $A' = S_k^-$. The proposition is proven.

Comment. When we have several constraints, the set of all the values that satisfy all these constraints is an intersection of the sets corresponding to individual constraints. If we were simply interested in when the set is compact or not, we could reduce the case of several constraints to problem to a simpler case of a single constraint, by using a known fact that the intersection of compact sets is a compact set. However, a similar reduction is not possible for *computable* compact sets, because, as will see in a second, there is no algorithm that would take two computable compact sets and produce an algorithm showing that their intersection is a computable compact set. Indeed, let us take $A = \{0, 1\}$ and $A_\varepsilon = \{0, 1 + \varepsilon\}$ for some small ε. The corresponding function $\varepsilon \to A \cap A_\varepsilon$ is discontinuous and thus, not computable: indeed, its value is equal to $\{0\}$ when $\varepsilon \neq 0$ and to $\{0, 1\}$ when $\varepsilon = 0$.

General case. In the general case, when we have both equalities and inequalities, we have a similar – but slightly more complex-to-formulate – result. Let us recall that a *modulus of continuity* of a function $F(x)$ is a function $\delta(\varepsilon)$ that, given a positive real number $\varepsilon > 0$, produces a positive real number $\delta > 0$ for which $d(x, x') \leq \delta$ implies $d(F(x), F(x')) \leq \varepsilon$.

Proposition 7. *Let $f_1, \ldots, f_n, g_1, \ldots, g_m$ be computable functions from a computable compact set X to real numbers. Let*

$$[\underline{c}_1, \overline{c}_1], \ldots, [\underline{c}_n, \overline{c}_n], [\underline{v}_1, \overline{v}_1], \ldots, [\underline{v}_m, \overline{v}_m]$$

be computable intervals for which the dependence of the set

$$\{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n, g_1(x) = v_1, \ldots, g_m(x) = v_m\}$$

on the parameters $c_1, \ldots, c_n, v_1, \ldots, v_m$ is continuous, with a known computable modulus of continuity. Then, there exists an algorithm that:

- *given computable values*

$$c_1 \in (\underline{c}_1, \overline{c}_1), \ldots, c_n \in (\underline{c}_n, \overline{c}_n), v_1 \in (\underline{v}_1, \overline{v}_1), \ldots, v_m \in (\underline{v}_m, \overline{v}_m),$$

- *computes the set*

$$\{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n, g_1(x) = v_1, \ldots, g_m(x) = v_m\}.$$

Proof. We want to prove that the desired level set

$$A \stackrel{\text{def}}{=} \{x : f_1(x) \geq c_1, \ldots, f_n(x) \geq c_n, g_1(x) = v_1, \ldots, g_m(x) = v_m\}$$

is a computable compact set. According to Proposition 4, to prove this, it is sufficient to provide an algorithm that, given a rational number $\varepsilon > 0$, produces a computable compact set A' for which $d_H(A', A) \leq \varepsilon$. In other words, it is sufficient to be able to compute, for each ε, an ε-approximation to the desired level set A.

Indeed, similarly to the proof of Proposition 6, each inequality $f_i(x) \geq c_i$ is equivalent to the condition that difference between the left-hand and the right-hand sides of this inequality is non-negative: $f_i(x) - c_i \geq 0$. Each equality $g_i(x) = v_i$ is equivalent to two inequalities: $g_i(x) \geq v_i$ and $v_i \geq g_i(x)$, i.e., to $g_i(x) - v_i \geq 0$ and $v_i - g_i(x) \geq 0$. Thus, all $n + m$ original constraints is satisfied if and only if the smallest of these differences is non-negative, i.e., when $f(x) \geq 0$, where we denoted

$$f(x) \stackrel{\text{def}}{=} \min(f_1(x) - c_1, \ldots, f_n(x) - c_n,$$

$$g_1(x) - v_1, v_1 - g_1(x), \ldots, g_m(x) - v_m, v_m - g_m(x)).$$

So, $A = \{x : f(x) \geq 0\}$. Since the functions $f_i(x)$ and $g_j(x)$ and the values c_i and v_j are computable, the function $f(x)$ is computable as well.

Since the modulus of continuity $\delta(\varepsilon)$ corresponding to the dependence of the solution set on the parameters is computable, we can compute the value δ for which if all the parameters are δ-close, the resulting solution sets are ε-close.

By the known property of a computable compact, for every rational number $\delta > 0$, we can algorithmically find the value $-\delta_- \in (-\delta, 0)$ for which the level set $S^- \stackrel{\text{def}}{=} \{x : f(x) \geq -\delta_-\}$ is a computable compact.

In terms of the original functions $f_i(x)$ and $g_j(x)$, this level set takes the form

$$S^- = \{x : f_1(x) \geq c_1 - \delta_-, \ldots, f_n(x) \geq c_n - \delta_-,$$

$$g_1(x) - v_1 \geq -\delta_-, v_1 - g_1(x) \geq -\delta_-, \ldots, g_m(x) - v_m \geq -\delta_-, v_m - g_m(x) \geq -\delta_-\}.$$

One can easily show that for every j, the inequalities $g_j(x) - v_j \geq -\delta_-$ and $v_j - g_j(x) \geq -\delta_-$ are equivalent to the double inequality $v_j - \delta_- \leq g_j(x) \leq v_j + \delta_-$. Thus,

$$S^- = \{x : f_1(x) \geq c_1 - \delta_-, \ldots, f_n(x) \geq c_n - \delta_-,$$

$$v_1 - \delta_- \leq g_1(x) \leq v_1 + \delta_-, \ldots, v_m - \delta_m \leq g_m(x) \leq v_m + \delta_-\}.$$

By considering all possible values $v_j' \in [v_j - \delta_-, v_j + \delta_-]$, we conclude that this set S^- is a union of the sets

$$S_{v'} = \{x : f_1(x) \geq c_1 - \delta_-, \ldots, f_n(x) \geq c_n - \delta_-, g_1(x) = v_1', \ldots, g_m(x) = v_m'\}$$

corresponding to all possible combinations of values $v_j' \in [v_j - \delta_-, v_j + \delta_-]$. For each of the sets $S_{v'}$, the values $c_i - \delta_-$ and v_j' are δ-close to the values c_i and v_j. Thus, by the choice of δ, each of the sets $S_{v'}$ is ε-close to the desired set A (in the sense of Hausdorff metric). Thus, by Proposition 2, the union S^- of the sets $S_{v'}$ is also ε-close to the desired set A. So, we have computed, for each $\varepsilon > 0$, a computable set S^- which is ε-close to A. According to Proposition 4, this implies that the set A itself is computable. The proposition is proven.

Comment. The above proof provides an algorithm, but does not provide us with any upper bound on the number of computation steps. In constructive mathematics, such algorithms are called *based on Markov principle* (see, e.g., [12, 20]) – after a principle formulated by A. A. Markov that if it is not true that an algorithm does not halt, then we conclude that this algorithm halts (even if we do not have any upper bound on the time that it takes for this algorithm to finish its computations). This is worth mentioning since there exist versions of constructive mathematics that only allow algorithms with explicitly computable bounds on computation time.

Optimization under constraints. In the previous text, we showed that the first problem – of computing the set S of all the values x that satisfy given constraints – is computable if the dependence is continuous. What about the second problem – of computing the maximum (or a minimum) of a computable function $f(x)$ over the set S (i.e., under the given constraints)? It turns out that this problem is computable too – because:

- computability of the solution set S means that this set S is a computable compact set, and
- as we have mentioned, there is an algorithm that computes the maximum and the minimum of a given computable continuous function over a computable compact S.

Comment: possible application to processing "fuzzy" (imprecise) data. The results about optimization under constraints may be helpful in processing *fuzzy data*, a known efficient way of representing imprecise expert knowledge; see, e.g., [9, 16]. The meed for special techniques for processing such expert knowledge comes from the fact that experts often express their estimates of the values of different quantities not by exact numbers, but by using words from natural language such as "small", "the size of an apple", etc. One way to describe this knowledge in terms understandable by a computer is to assign, to each possible value x of the corresponding quantity, a number $\mu(x)$ from the interval $[0, 1]$ describing to what extent the experts believe that this value is, e.g., small. This degree can be elicited from the experts – or estimated as a ratio of experts who believe that x is small to the total number of interviewed experts. The function $\mu(x)$ that describes how this degree depends on the value x is called a *membership function*.

One of the reasons why we may ask the experts to estimate the values of the quantities $x_1, \ldots, x_n$ is that we may want to use the values of these easier-to-estimate quantities estimates to find the value of a related more-difficult-to-estimate quantity y. For that, we need to know the relation between x_i and y, i.e., we need to know a data processing algorithm $y = f(x_1, \ldots, x_n)$ that transforms the values of x_i into the desired estimate for y. Once we know this algorithms, and we know the membership functions $\mu_i(x_i)$ corresponding to the inputs, we need to estimate the resulting membership function $\mu(y)$. Reasonable

arguments (see, e.g., [10]) lead to the following formula for $\mu(y)$:

$$\mu(y) = \max\{\min(\mu_1(x_1), \ldots, \mu_n(x_n)) : f(x_1, \ldots, x_n) = y\}.$$

In other words, estimating the value $\mu(y)$ requires that we find the maximum of a computable function $\min(\mu_1(x_1), \ldots, \mu_n(x_n))$ under a constraint $f(x_1, \ldots, x_n) = y$. Thus, general algorithms for optimization under constraints can be used for processing imprecise ("fuzzy") data.

Finding optimal solutions under constraints. In some practical situations, in addition to finding the optimal value v of a function $f(x)$ under given constraints, it is also desirable to find the values x for which this optimal is attained. For example, in decision making, we are not just interested in knowing how good a situation can be if we make an appropriate choice x, we also want to know exactly what alternative x we should choose.

If we describe this requirement literally, this would mean that we are looking for the set $\{x : f(x) = v\}$. In practice, a very small deviation from the optimum does not change anything – especially since the function $f(x)$ that describes the dependence of the benefits on the alternative x is usually only known approximately anyway. Thus, from the practical viewpoint, it makes more sense to consider a set $\{x : f(x) \geq v - \varepsilon\}$ for some small $\varepsilon > 0$. This value we can already compute – provided that the dependence of the level set on ε is continuous.

Remaining open problems. In this paper, we consider "unconditional" constraints of the type $f(x) \geq c$, $f(x) \leq c$, and $f(x) = c$. In some applications, we have "conditional" constraints, i.e., constraints that only hold under certain conditions. For example, we can have constraints of the type "if $f(x) \geq c$, then $g(x) = v$". It would be nice to extend our computability results to such conditional constraints.

Acknowledgments. This work was supported in part by the National Science Foundation grants HRD-0734825 and DUE-0926721 and by Grant 1 T36 GM078000-01 from the National Institutes of Health.

References

[1] O. Aberth, *Precise Numerical Analysis Using C++*, Academic Press, New York, 1998.

[2] S. Basu, R. Pollack, and M.-F. Roy, *Algorithms in Real Algebraic Geometry*, Springer-Verlag, Berlin, 2006.

[3] M. J. Beeson, *Foundations of constructive mathematics*, Springer-Verlag, N.Y., 1985.

[4] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, 1967.

[5] E. Bishop and D. S. Bridges, *Constructive Analysis*, Springer, N.Y., 1985.

[6] D.S. Bridges, *Constructive Functional Analysis*, Pitman, London, 1979.

[7] D. S. Bridges and S. L. Via, *Techniques of Constructive Analysis*, Springer-Verlag, New York, 2006.

[8] J. L. Kelley, *General Topology*, Springer Verlag, Berlin-Heidelberg-New York, 1975.

[9] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[10] V. Kreinovich, "Relation between interval computing and soft computing", In: C. Hu, R. B. Kearfott, A. de Korvin, and V. Kreinovich (eds.), *Knowledge Processing with Interval and Soft Computing*, Springer Verlag, London, 2008, pp. 75–97.

[11] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1998.

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

[13] Y. Matsumoto, *An Introduction to Morse Theory*, American Mathematical Society, Providence, Rhode Island, 2001.

[14] J. Milnor, *Morse Theory*, Princeton University Press, Princeton, New Jersey, 1963.

[15] B. Mishra, "Computational real algebraic geometry", in: *Handbook on Discreet and Computational Geometry*, CRC Press, Boca Raton, Florida, 1997.

[16] H. T. Nguyen and E. A. Walker, *First Course In Fuzzy Logic*, CRC Press, Boca Raton, Florida, 2006.

[17] L. Nicolaescu, *An Invitation to Morse Theory*, Springer Verlag, New York, 2007.

[18] M. B. Pour-El and J. I. Richards, *Computability in Analysis and Physics*, Springer, Berlin, 1989.

[19] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, 2nd ed., Berkeley and Los Angeles, 1951, 63 pp.

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

[21] S. Willard, *General Topology*, Dover Publ., New York, 2004.