Computational Complexity of Crude Range Estimation and Fuzzy Optimization

G. William Walster[1] and Vladik Kreinovich[2]

[1]Interval Technology Engineering Manager
Sun Microsystems, Inc.
16 Network Circle, MS UMPK16-304
Menlo Park, CA 94025
bill.walster@eng.sun.com

[2]Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@cs.utep.edu

Abstract

It is often important to check whether the maximum $\max_B f$ of a given function f is smaller than the current lower bound C for the global maximum. Empirical evidence shows that different instances of this checking problem have different relative complexity: the larger the difference $C - \max f$, the easier the problem. In general, the fewer global maxima, the easier the problem; and finally, the further away global maxima from each other, the easier the problem. It is difficult to formalize this empirical difference in complexity in standard complexity theory terms, because all these cases are NP-hard. In this paper, we use the analysis of mathematical optimization problems emerging from fuzzy optimization to propose a new "robust" formalization of relative complexity which takes into consideration numerical inaccuracy. This new formalization enables us to theoretically explain the empirical results on relative complexity.

1 Crude Approximate Range Estimation Is a Crucial Step in Solving Many Important Real-Life Problems

In many real-life situations, we want to find the *best* decision, the *best* control strategy, etc. The corresponding problems are naturally formalized as *optimization* problems: we have a function $f(x_1, \ldots, x_n)$ of several variables, and we want to find the values $(x_1, \ldots, x_n)$ for which this function attains the largest (or the smallest) possible value.

Many numerical algorithms have been proposed for solving optimization problems. Unfortunately, many of these algorithms often end up in a *local* maximum instead of the desired global one.

- In some practical situations, e.g., in decision making, the use of local maximum simply degrades the quality of the decision but is not, by itself, disastrous.
- However, in some other practical situations, missing a global maximum or minimum may be disastrous.

Let us give two example:

- In *chemical engineering*, global minima of the energy function often describe the stable states of the system. If we miss such a global minimum, the chemical reactor may go into an unexpected state, with possible serious consequences.
- In *bioinformatics*, the actual shape of a protein corresponds to the global minimum of the energy function. If we find a local minimum instead, we end up with a wrong protein geometry. As a result, if we use this wrong geometry as a computer simulation for testing recommendations on the medical use of chemicals, we may end up with medical recommendations which harm a patient instead of curing him.

For such applications, it is desirable to use *rigorous*, *automatically verified* methods of global optimization, i.e., methods which never discard an actual global maximum; for a survey of such methods, see, e.g., [6]. These methods usually start with a large "box" on which a function is defined (and on which global maxima can be located), and produce a list of small-size boxes with the property that every global maximum is guaranteed to be contained in one of these boxes.

Most of such guaranteed methods use (a version of) *interval computations*. The main idea of interval computations is as follows: To solve a given numerical problem (e.g., an optimization problem), numerical methods typically generate better and better estimates for different quantities related to the problem – such as the actual global maximum of the function, the value of its partial derivatives at different points, etc.

- In *traditional* numerical techniques, for each approximated numerical quantity, an approximation is a real number, with no guarantees on the approximation accuracy.
- In *interval* methods, at any given moment of time, for each approximated quantity x, we compute not only the approximate value $\widetilde{x}$, but also the *upper bound* Δ on the possible approximation error, i.e., a number Δ for which we are guaranteed that $|x - \widetilde{x}| \le \Delta$. In other words, at any step, we have not only an approximate value $\widetilde{x}$ of the approximate quantity, we also have an *interval* $\mathbf{x} = [\widetilde{x} - \Delta, \widetilde{x} + \Delta]$ which is *guaranteed* to contain the (unknown) actual value of x.

As we have mentioned, rigorous methods of global optimization start with a large box as a location of the unknown global maxima and gradually replace it will a small finite collection of small boxes. The decrease in a box size is usually achieved by dividing one of the boxes into several sub-boxes and eliminating some of these sub-boxes.

When can we eliminate a sub-box B? At every stage of the optimization algorithm, we have already computed several values of the optimized function $f(x_1, \ldots, x_n)$, so we know that the global maximum of the function f cannot be smaller than the largest M of these already computed values. Thus, if we can guarantee that the maximum of the function f on a box B is smaller than M, we can thus exclude this box from the list of possible locations of a global maximum.

This idea would not work efficiently if we had to actually compute the exact range of a function f on each subbox: this would require a lot of computation time. Luckily, for the desired exclusion of subboxes, we do not need to know the *exact* range of f on B (i.e., the exact values of the maximum and the minimum of f on B); for most subboxes, this range is far from the global maximum, so it is sufficient to check whether the maximum is $< M$, i.e., to produce crude *approximate* estimates of this range. Thus, *approximate range estimation is a crucial step in solving many important real-life problems.* This problem is mentioned as the first major problem in the keynote talk of the recent biannual international conference on interval computations and validated numerics [19].

There are other cases when a crude range estimation is important. Let us give three such examples:

- There are many cases when it is (relatively) easy to estimate the range: e.g., when a function is monotonic in each of the variables. How can we check this monotonicity? A function f is, e.g., increasing in x_1 if the partial derivative $\dfrac{\partial f}{\partial x_i}$ is positive for all the values $(x_1, \ldots, x_n)$ from a box B. To check this property, we must confirm that the minimum of this derivative on B is positive. Again, we do not need to evaluate the exact range for this derivative, all we need is to check whether the lower endpoint for this range is positive. In other words, all we need is a crude approximate estimate for this range.

- Similarly, when the algorithm computing the function $f(x_1, \ldots, x_n)$ contains branching over the sign of some quantity $g(x_1, \ldots, x_n)$, then we can often simplify the computations of f on a box B if we know that for values from B, only one of the branches is actually used: e.g., if $g(x_1, \ldots, x_n) > 0$ for all $(x_1, \ldots, x_n) \in B$.

- Optimization is just one example of the importance of crude estimates. In some real-life problems, we are not yet ready for optimization, e.g., because the problem has so many constraints that even finding *some* values $x = (x_1, \ldots, x_n)$ of the parameters x_i which satisfy all these constraints is an extremely difficult task. For such problems, we arrive at the problem of satisfying given constraints, e.g., solving a given system of equations. For such problems, we can use similar interval techniques to get a small finite set of small boxes containing solutions, and crude range estimation is an important part of these techniques.

2 Computational Complexity Results as a Guidance for Practical Algorithm Design

Checking whether $\max f < C$ is a crucial step in solving real-life problems. It is therefore important to have a good algorithm for solving this problem. Can we design a universal efficient algorithm for solving it, an algorithm which would always produce a correct answer in reasonable computation time? It looks like we cannot, because it has been shown that this problem is NP-hard (see, e.g., [18]). For those readers who are not familiar with the precise definition of NP-hardness, we can simply explain that it means exactly that such a universal effective algorithm is highly unlikely.

Since we cannot have an algorithm which *always* works, we must concentrate on the next best thing: on finding the classes of problems for which efficient algorithms are possible, and on providing efficient algorithms for these classes. Usually, such classes are formulated in terms of some appropriate parameters.

In finding appropriate parameters, we are not completely clueless. People have been solving optimization problems for quite some time, so there is an experience and intuition which tells us which problems are more difficult and which problems are easier. For example, it is usually true that the larger the difference between the desired bound C and the actual maximum $\max f$, the easier the problem. In general:

- the complexity of locating global maxima of f is empirically known to depend on the number of these global maxima: the fewer global maxima, the easier the problem;
- also, when we compare problems with the same number of global maxima, problems in which these maxima are well separated are usually easier to solve while problems in which these maxima are close are more difficult.

This intuition seems to be supported by the experience of numerical computations, so we would like to use it as a guidance in designing new algorithms. To make this guidance more convincing, it is desirable to confirm this empirical intuition about the relative complexity of different problems by some precise complexity results.

The need for this confirmation comes not from any *negative* feeling of mistrust of experts, no, it comes from a very *positive* history of innovations in numerical mathematics – where many successful innovations come from ideas which were originally contrary to the prevailing intuition (this is why they are called innovations).

3 How Can We Describe the Desired Complexity: The Problem

A traditional approach to estimating complexity of a computational problem is to see if this problem is NP-hard – or it can be solved by a feasible algorithm. Unfortunately, this traditional approach does not work here, because the problem of computing the maximum $\max f$ on a given box with a given accuracy ε is NP-hard for an arbitrary ε, large or small [18]. We therefore need a different approach to comparing complexity of different cases of this general problem.

4 Case Study Which Helps Us Solve This Problem: Mathematical Optimization Problems Emerging from Fuzzy Optimization

In many real-life problems, we know the exact form of the objective function $f(x)$, but the set B over which we optimize is fuzzy.

For example, when an automobile company designs a luxury object such as a "flashy" sports car, its goal is to maximize the profit. Within a reasonable sales prediction model, profit is a well-defined function, but "flashiness" is clearly a fuzzy notion.

In general, we have a problem of maximizing a (crisp) function $f(x)$ over a fuzzy set B characterized by a membership function $\mu_B(x)$. In their 1970 paper [2], Bellman and Zadeh proposed to describe the degree $\mu_M(x)$ to which a given element x is a solution to this fuzzy optimization problem as a degree to which B is true *and* x maximizes f. There are several ways to describe this degree in terms of $f(x)$ and $\mu_B(x)$ (see, e.g., [7]), e.g., as

$$\mu_M(x) = f_{\&}\left(\mu_B(x), \frac{f(x)-m}{M-m}\right),$$

where $f_{\&}(a,b)$ is a t-norm, and m and M are, correspondingly, the global minimum and the global maximum of the function $f(x)$ on the universe of discourse X.

If we want to select a single design, then it is natural to select x for which this degree is the largest: $\mu_M(x) \to \max_X$. Thus, the original fuzzy optimization problem leads to a crisp mathematical optimization problem with a new objective function $\mu_M(x)$.

At first, we have one more example of a mathematical (crisp) optimization problem. However, if we look at the new objective function more attentively, we will see that there is a principal difference between the crisply-formulated optimization problems and the crisp optimization problems resulting from fuzzy optimization:

- In a crisply formulated optimization problem, the objective function $f(x)$ is fixed.
- On the other hand, in a crisp optimization problems resulting from fuzzy optimization, the corresponding objective function $\mu_M(x)$ is not that fixed.

Indeed, for every x, the numerical value of this new objective function depends on the numerical value of the membership function $\mu_B(x)$. This membership value, in its turn, comes from an expert and cannot be very precise. If we use a slightly different elicitation technique, we may end up with a slightly different value of $\mu_B(x)$ and thus, a slightly different value of $\mu_M(x)$.

Thus, the same real-life fuzzy optimization problem can lead not only to the objective function $\mu_M(x)$, but also to other objective functions which are close to the original function $\mu_M(x)$.

It is therefore reasonable to require that the algorithms not only work on a given function f, but that they work *robustly* in the sense that they produce a correct answer not only for the exact given function f, but for all the functions $\widetilde{f}$ which are sufficiently close to this f.

In some cases, the values $\mu_B(x)$ do not come directly from elicitation. For example, if we are looking for the optimal control of a system described by fuzzy conditions (corresponding to the set B), then the set B is described by if-then rules $A_i(x) \to B_i(y)$ with fuzzy premises and fuzzy conclusions. In this case, since the premises $A_i(x)$ are also fuzzy, not only the value of $\mu_B(x)$ is not precisely known, but we are also not sure to which exactly value of x the given value of μ_B refers to: it may refer to the given x, and it may as well refer to some value $\widetilde{x}$ which is close to x.

In view of this possibility, in this paper, we will consider algorithms which are "robust" in the sense that they are applicable not only to the original function f, but also to close functions $\widetilde{f}$, and we will consider two types of closeness:

- first, a more natural *y-closeness* which means that for every input x_i, the y-values, i.e., the values of $f(x_1, \ldots, x_n)$ and $\widetilde{f}(x_1, \ldots, x_n)$, are sufficiently close;
- second, an (also needed) *x-closeness*, which takes into consideration the fact that the inputs x_i to the function f are also not presented exactly.

5 In Hindsight, This New Approach to Computational Complexity Makes Perfect Sense Even Without Fuzzy

One of the main reasons why traditional complexity approach is not exactly applicable here is that traditional complexity theory was originally designed for *discrete* problems, for which the answer is either correct or not. In contrast, we are interested in a *continuous* problem, in which the answer is correct to a certain *accuracy*. Similarly, the input to the problem (i.e., the optimized function f) is not given exactly, it is given (due to rounding errors etc.) only with a certain accuracy. Thus, when we feed a function f to the algorithm, the actual function $\widetilde{f}$ may be slightly different from f.

Thus, it makes perfect sense to consider algorithms which are applicable not only to the original objective function f, but also to all objective functions which are sufficiently close to f.

6 The Main Result: The Larger the Difference $C - \max f$, the Easier the Problem

In order to formulate this result, we must recall some basic definitions of computable ("constructive") real numbers and computable functions from real numbers to real numbers (see, e.g., [1, 3, 4, 5, 18]):

Definition 1. *A real number x is called computable if there exists an algorithm (program) that transforms an arbitrary integer k into a rational number x_k that is 2^{-k}–close to x. It is said that this algorithm computes the real number x.*

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

Definition 2. *A function $f(x_1, \ldots, x_n)$ from real numbers to real numbers is called computable if there exist algorithms U_f and φ, where:*

- *U_f is a rational-to-rational algorithm which provides, for given rational numbers $r_1, \ldots, r_n$ and an integer k, a rational number $U_f(r_1, \ldots, r_n, k)$ which is 2^{-k}-close to the real number $f(r_1, \ldots, r_n)$, and*

  $$|U_f(r_1, \ldots, r_n, k) - f(r_1, \ldots, r_n)| \le 2^{-k},$$

  and

- *φ is an integer-to-integer algorithm which gives, for every positive integer k, an integer $\varphi(k)$ for which $|x_1 - x_1'| \le 2^{-\varphi(k)}$, ..., $|x_n - x_n'| \le 2^{-\varphi(k)}$ implies that*

  $$|f(x_1, \ldots, x_n) - f(x_1', \ldots, x_n')| \le 2^{-k}.$$

When we say that a computable function is given, we mean that we are given the corresponding algorithms U_f and φ.

Let us start with the analysis of non-robust algorithms for checking whether $\max f < C$.

Definition 3. *By an algorithm for checking whether* $\max f < C$ *(or simply checking algorithm, for short), we mean an algorithm U which takes as input a triple (B, f, C), where:*

- *B is a computable box,*
- *f is a computable function on the box B, and*
- *C is a computable real number,*

such that:

- *if the algorithm U returns "yes", then $\max\limits_B f < C$; and*
- *if the algorithm U returns "no", then $\max\limits_B f \ge C$.*

In this definition, we did not require that U always returns "yes" or "no"; we allow this algorithm to sometimes return "do not know" (or simply stall without returning any answer). The reason for this is that no checking algorithm can always return "yes" or "no":

Proposition 1. *No algorithm is possible which, given a computable function f on a computable box B and a computable real number C, checks whether* $\max f < C$.

(For the reader's convenience, all the proofs are placed in the special – last – Proofs section.)

If we know the lower bound for the difference $C - \max f$, then such an algorithm is already possible:

Proposition 2. *Let $D > 0$ be a computable real number. Then, there exists a checking algorithm U_D which is applicable to all functions f for which $C - \max f > D$.*

The meaning of this proposition is reasonably straightforward:

- According to Proposition 1, if we require that an algorithm's answer to the question "$\max f < C$?" is always correct, then this algorithm *cannot* be *always* applicable, there will always be cases for which this algorithm fails to produce any answer (positive or negative).

- Proposition 2 says that, by an appropriate choice of an algorithm, we can restrict the cases when an algorithm refuses to answer to situations in which the difference $C - \max f$ is small ($\le D$); for situations in which this difference is large enough, the above-mentioned algorithm produces a definite (and correct) answer.

Proposition 2 does not distinguish between the classes of problems corresponding to different values of D. To make this distinction, we must look for *robust* algorithms instead of simply algorithms which work for exact data. Let us start with a definition of robustness.

Definition 4. *Let $\varepsilon > 0$ be a real number.*

- *We say that two functions $f(x_1, \ldots, x_n)$ and $\widetilde{f}(x_1, \ldots, x_n)$ are ε-y-close if for every input $(x_1, \ldots, x_n)$, their values are ε-close:*

$$|f(x_1, \ldots, x_n) - \widetilde{f}(x_1, \ldots, x_n) \le \varepsilon.$$

- *We say that an algorithm for checking whether $\max f < C$ is ε-y-robustly applicable to the input (B, f, C), if it is applicable not only for this function f, but also for an input $(B, \widetilde{f}, C)$ for an arbitrary function $\widetilde{f}$ which is ε-y-close to f.*

Theorem 1. *Let $D > 0$ be a computable real number, and let $\varepsilon > 0$ be another computable real number. Then:*

- *If $\varepsilon < D$, there exists a checking algorithm which is ε-y-robustly applicable to all functions f for which $C - \max f > D$.*

- *If $\varepsilon > D$, then no checking algorithm which is ε-y-robustly applicable to all functions f for which $C - \max f > D$.*

This result shows that the larger the difference $C - \max f$, the easier it is to check that $\max f < C$. Indeed, let $D_1 < D_2$; let us take $D = (D_1 + D_2)/2$. Then, according to Theorem 1:

- there exists a checking algorithm which is D-y-robustly applicable to all functions f for which $C - \max f > D_2$; and
- no checking algorithm is possible which is D-y-robustly applicable to all functions f for which $C - \max f > D_1$.

In other words, if $D_1 < D_2$, then the checking problem corresponding to D_2 is indeed easier to solve.

7 Second Result: The Fewer Global Maxima, the Easier the Problem

In [18], we provided a partial justification of this intuition. Namely, the following two results are true:

Theorem [8, 9, 12, 14]. *There exists an algorithm U such that:*

- *U is applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ that attains its maximum on a computable box $B = [a_1, b_1] \times \ldots \times [a_n, b_n]$ at exactly one point $x = (x_1, \ldots, x_n)$,*
- *for every such function f, the algorithm U computes the global maximum point x.*

Theorem [11, 12, 13, 14, 15, 16, 17, 18]. *No algorithm U is possible such that:*

- *U is applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ that attains its maximum on a computable box $B = [a_1, b_1] \times \ldots \times [a_n, b_n]$ at exactly two points, and*
- *for every such function f, the algorithm U computes one of the corresponding global maximum points x.*

These results *partially* explain the above intuition because they show that the problem of locating global maxima is easier if we have a single global maximum and more difficult if we have several global maxima. These results, however, *do not completely* explain this intuition because they do not explain why, say, a problem with three global maxima is more complex than a problem with two global maxima.

Similar results hold for roots (solutions) of a system of equations:

Definition 5. *By a computable system of equations we mean a system $f_1(x_1, \ldots, x_n) = 0, \ldots, f_k(x_1, \ldots, x_n) = 0$, where each of the functions f_i is a computable function on a computable box $B = [a_1, b_1] \times \ldots \times [a_n, b_n]$.*

Theorem [8, 9, 12, 14]**.** *There exists an algorithm U such that:*

- *U is applicable to an arbitrary computable system of equations which has exactly one solution, and*
- *for every such system of equations, the algorithm U computes its solution.*

Theorem [11, 12, 13, 14, 15, 16, 17, 18]**.** *No algorithm U is possible such that:*

- *U is applicable to an arbitrary computable system of equations which has exactly two solutions, and*
- *for every such system of equations, the algorithm U computes one of its solutions.*

8 Third Result: The Closer the Maxima, the More Difficult the Problem

Definition 7. *By a global optimization algorithm, we mean an algorithm which (whenever it is applicable) returns the list of locations of all global maxima.*

Definition 8. *Let $d > 0$. We say that points $x^{(1)}, \ldots, x^{(m)}$ are d-separated if the distance between every two different points from this list is $\geq d$.*

Theorem [8, 9, 12, 14]**.** *Let m be a given integer, and $d > 0$ be a computable real number. Then, there exists an optimization algorithm U such which is applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ which attains its maximum on a computable box B at exactly m d-separated points.*

This result shows that if we know the lower bound on the distance between the global maxima, then the optimization problem becomes easier. This result by itself, however, does not explain why the closer the maxima, the more complex the optimization problem seems to get. To explain this empirical fact, we will again use a notion of robustness.

Definition 6. *Let $\delta > 0$ be a real number.*

- *We say that a 1-1 mapping $R^n \to R^n$ is a δ-isometry if T changes the distance $\rho(x, x')$ between every two points $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$ by $\le \delta$, i.e., for for every two points x and x', we have*

$$|\rho(x, x') - \rho(Tx, Tx')| \le \delta.$$

- *We say that two functions $f(x_1, \ldots, x_n)$ and $\widetilde{f}(x_1, \ldots, x_n)$ are δ-x-close if there exists a δ-isometry T for which $\widetilde{f}(x) = f(Tx)$.*

- *We say that an algorithm is δ-x-robustly applicable to the input f, if it is applicable not only for this function f, but also for an arbitrary function $\widetilde{f}$ which is δ-x-close to f.*

Theorem 2. *Let $d > 0$ be a computable real number, and let $\delta > 0$ be another computable real number. Then:*

- *If $\delta < d$, there exists an optimization algorithm U which is δ-x-robustly applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ which attains its maximum on a computable box B at exactly m d-separated points.*

- *If $\delta > d$, then no optimization algorithm U can be δ-x-robustly applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ which attains its maximum on a computable box B at exactly m d-separated points.*

This result shows that the larger the lower bound d between the global maxima, the easier it is to solve the optimization problem. Indeed, let $d_1 < d_2$; let us take $d = (d_1 + d_2)/2$. Then, according to Theorem 1:

- there exists an optimization algorithm which is d-x-robustly applicable to all functions f for which global maxima are d_2-separated; and

- no optimization algorithm is possible which is d-x-robustly applicable to all functions f for which global maxima are d_1-separated.

In other words, if $d_1 < d_2$, then the optimization problem corresponding to d_2 is indeed easier to solve.

Similar results hold for roots (solutions) of a system of equations:

Definition 9. *By a system solving* algorithm, *we mean an algorithm which (whenever it is applicable) returns the list of solutions to a given computable system of equations.*

Theorem [8, 9, 12, 14]**.** *Let m be a given integer, and $d > 0$ be a computable real number. Then, there exists a system solving algorithm U such which is applicable to an arbitrary computable computable system of equations which has exactly m d-separated solutions.*

Definition 6′. *Let $\delta > 0$ be a real number.*

- *We say that two systems of equations*

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_k(x_1, \ldots, x_n) = 0,$$

  and

$$\widetilde{f}_1(x_1, \ldots, x_n) = 0, \ldots, \widetilde{f}_k(x_1, \ldots, x_n) = 0$$

  are δ-x-close if there exists a δ-isometry T for which $\widetilde{f}_i(x) = f_i(Tx)$ for all $i = 1, \ldots, k$.

- *We say that an algorithm is δ-x-robustly applicable to the system $f_1 = 0, \ldots, f_k = 0$, if it is applicable not only for this system, but also for an arbitrary systems of equations $\widetilde{f}_1 = 0, \ldots, \widetilde{f}_k = 0$ which is δ-x-close to the system $f_1 = 0, \ldots, f_k = 0$.*

Theorem 2′. *Let $d > 0$ be a computable real number, and let $\delta > 0$ be another computable real number. Then:*

- *If $\delta < d$, there exists a system solving algorithm U which is δ-x-robustly applicable to an arbitrary computable system of equations which has exactly m d-separated solutions.*

- *If $\delta > d$, then no system solving algorithm U can be δ-x-robustly applicable to an arbitrary computable system of equations which has exactly m d-separated solutions.*

9 Proofs

9.1 Proof of Proposition 1

It is easy to show that a constant function $f(x_1, \ldots, x_n) \equiv 0$ is a computable function. For this function, $\max f = 0$. Thus, if we had an algorithm which checks, given B, f, and C, whether $\max f < C$ or not, then we will be able to check whether $C > 0$ for a given computable real number C. However, it is known that it is algorithmically impossible to check whether a given computable real number is positive or not [1, 3, 4, 5, 10, 18]). Thus, a checking algorithm cannot be always applicable. The proposition is proven.

9.2 Proof of Proposition 2

1. It is known that there exists an algorithm which, given a computable function on a computable box, and a given $\delta > 0$ returns a rational number M which is δ-close to $\max f$ [1, 3, 4, 5, 18]. Let us reproduce the main idea of this proof.

1.1. First, we prove that there exists an integer m for which the 2^{-m}-approximation δ_m to δ exceeds $3 \cdot 2^{-m}$.

Indeed, since $\delta > 0$, we have $\delta > 2^{-k}$ for some k. Therefore, for the $2^{-(k+2)}$-approximation δ_{k+2} to δ, we get $|\delta_{k+2} - \delta| \le 2^{-(k+2)}$ hence

$$\delta_{k+2} \ge \delta - 2^{-(k+2)} > 2^{-k} - 2^{-(k+2)} = 3 \cdot 2^{-(k+2)}.$$

So, the existence is proven for $m = k + 2$.

This m can be algorithmically computed as follows: we sequentially try $m = 0, 1, 2, \ldots$ and check whether $\delta_m > 3 \cdot 2^{-m}$; when we get the desired inequality, we stop.

1.2. Let us now show that for the integer m computed according to Part 1.1 of this proof, we have $\delta > 2 \cdot 2^{-m}$.

Indeed, since $\delta_m > 3 \cdot 2^{-m}$ and $|\delta - \delta_m| \le 2^{-m}$, we can conclude that

$$\delta \ge \delta_m - 2^{-m} > 3 \cdot 2^{-m} - 2^{-m} = 2 \cdot 2^{-m}.$$

So, if we can find a rational number M which is $2 \cdot 2^{-m}$-close to $\max f$, this rational number will thus be also δ-close to $\max f$.

1.3. Let us now use this m to compute the desired δ-approximation to $\max f$.

1.3.1. By using the second algorithm φ in the definition of a computable function, we can find a value $\varphi(m)$ such that if $|x_i - x_i'| \le \varphi(m)$ for all $i = 1, \ldots, n$, then

$$|f(x_1, \ldots, x_n) - f(x_1', \ldots, x_n')| \le 2^{-m}.$$

For each dimension $[a_i, b_i]$ of the box B, we can then take finitely many values

$$r_i^{(1)}, r_i^{(2)} = r_i^{(1)} + \varphi(m), r_i^{(3)} = r_i^{(2)} + \varphi(m), \ldots, r_i^{(N_i)} = r_i^{(N_i-1)} + \varphi(m)$$

(separated by $\varphi(m)$) which cover the corresponding interval. Then, each value $x_i \in [a_i, b_i]$ will be different by one of these values $r_i^{(k_i)}$ by $\le \varphi(m)$.

1.3.2. Combining the values corresponding to different dimensions, we get a finite list of rational-valued vectors $\left(r_1^{(k_1)}, \ldots, r_n^{(k_n)}\right)$ with the property that every vector $(x_1, \ldots, x_n) \in B$ is $\varphi(m)$-close to one of these vectors.

Due to the definition of $\varphi(m)$, this means that each value $f(x_1, \ldots, x_n)$ is 2^{-m}-close to one of the values $f\left(r_1^{(k_1)}, \ldots, r_n^{(k_n)}\right)$. Therefore, the desired $\max f$ is 2^{-m}-close to the maximum of all the values $f\left(r_1^{(k_1)}, \ldots, r_n^{(k_n)}\right)$.

By using the algorithm U_f, we can compute each of these values with the accuracy 2^{-m}. Thus, the maximum M of thus computed rational values $U_f\left(r_1^{(k_1)}, \ldots, r_n^{(k_n)}, m\right)$. is 2^{-m}-close to the maximum of all the values $f\left(r_1^{(k_1)}, \ldots, r_n^{(k_n)}\right)$, and hence, $2 \cdot 2^{-m}$-close to $\max f$. Thus, M is indeed δ-close to $\max f$. The first part is proven.

2. The desired checking algorithm U_D can be therefore composed as follows:

- First, since $\delta = D/4$ is a computable number, we can use Part 1.1 of this proof to (constructively) find m for which
$$\delta = D/4 > 2 \cdot 2^{-m}. \tag{1}$$
- Then, we use Part 1 of this proof to compute a rational number M for which
$$|M - \max f| \leq 2 \cdot 2^{-m}. \tag{2}$$
- Third, we use the fact that C is a computable real number and generate the rational number C_{m-1} for which
$$|C - C_{m-1}| \leq 2^{-(m-1)} = 2 \cdot 2^{-m}. \tag{3}$$
- Finally, we check the inequality
$$C_{m-1} - M > 4 \cdot 2^{-m}. \tag{4}$$
If this inequality holds, we conclude that $\max f < C$.

To complete the proof, we must check two things:

- First, that the above checking algorithm is correct, i.e., that whenever this algorithm concludes that $\max f < C$, it is indeed true that $\max f < C$.
- Second, that the above checking algorithm U_D is indeed applicable to all functions f for which $C - \max f > D$.

3. Let us first prove that the above algorithm U_D is correct.

Indeed, if the inequality (4) holds, then $C_{m-1} > M + 4 \cdot 2^{-m}$. Using (3), we can then conclude that $C \geq C_{m-1} - 2 \cdot 2^{-m}$ hence
$$C \geq C_{m-1} - 2 \cdot 2^{-m} > M + 2 \cdot 2^{-m}.$$
Finally, from (2), we conclude that $M \geq \max f - 2 \cdot 2^{-m}$, hence
$$C > M + 2 \cdot 2^{-m} \geq \max f - 2 \cdot 2^{-m} + 2 \cdot 2^{-m} = \max f.$$
Correctness is proven.

4. Let us now complete our proof by showing that the above algorithm U_D is applicable to all functions f for which $C - \max f > D$.

Indeed, let $C - \max f > D$, i.e., that $C > \max f + D$. Due to formula (1), we have $D > 8 \cdot 2^{-m}$ hence
$$C > \max f + D > \max f + 8 \cdot 2^{-m}.$$

From (4), we can now conclude that

$$C_{m-1} \geq C - 2 \cdot 2^{-m} > \max f + 8 \cdot 2^{-m} - 2 \cdot 2^{-m} = \max f + 6 \cdot 2^{-m}.$$

From (2), we conclude that $\max f \geq M - 2 \cdot 2^{-m}$, hence

$$C_{m-1} > M - 2 \cdot 2^{-m} + 6 \cdot 2^{-m} = M + 4 \cdot 2^{-m},$$

i.e., the inequality (4) is indeed satisfied. Thus, for such a function f, the algorithm U_D will indeed return the correct answer.

The proposition is proven.

9.3 Proof of Theorem 1

1. Let us first show that if $\varepsilon < D$, then there exists a checking algorithm which is ε-y-robustly applicable to all functions f for which $C - \max f > D$.

Indeed, let us show that in this case, we can compute a computable positive real number $\widetilde{D} = D - \varepsilon$, and then use the (non-robust) checking algorithm $U_{\widetilde{D}}$ described in the proof of Proposition 2. Let us prove that this algorithm is indeed ε-y-robustly applicable to all functions f for which $C - \max f > D$.

By definition of robustness, we need to prove that the algoirithm $U_{\widetilde{D}}$ is applicable to every function $\widetilde{f}$ which is ε-close to a function f for which

$$C - \max f > D.$$

Indeed, when $\widetilde{f}$ is close to such a function f, we have $|\max \widetilde{f} - \max f| \leq \varepsilon$, hence $\max \widetilde{f} \leq \max f + \varepsilon$, and so

$$C - \max \widetilde{f} \geq C - \max f - \varepsilon > D - \varepsilon = \widetilde{D}.$$

Thus, by Proposition 2, the algorithm $U_{\widetilde{D}}$ is indeed applicable to the function $\widetilde{f}$. The statement is proven.

2. Let us now prove that if $\varepsilon > D$, then no checking algorithm U is possible which is ε-y-robustly applicable to all functions f for which $C - \max f > D$.

Indeed, if such an algorithm U was possible, we would be able to check whether a given computable real number α is positive or not, which, as we have already mentioned, is known to be impossible.

Since $\varepsilon > D$, the difference $D - \varepsilon$ is a computable negative real number, and hence, for every α, the number

$$C = \max\left(\alpha, \frac{D - \varepsilon}{2}\right) \geq \frac{D - \varepsilon}{2}$$

is also a computable real number. It is easy to check that $\alpha > 0$ if and only if $C > 0$, so, to check whether $\alpha > 0$, it is sufficient to be able to check whether $C > 0$ for all real numbers

$$C \geq \frac{D - \varepsilon}{2}. \tag{5}$$

To check this auxiliary inequality $C > 0$, we apply the hypothetic algorithm U to the constant-valued function $\widetilde{f}(x_1, \ldots, x_n) \equiv 0$ (for which $\max \widetilde{f} = 0$) and to this number C.

The algorithm U is applicable to this function $\widetilde{f}$ because of the following:

- The function $\widetilde{f}$ is ε-close to another constant-valued computable function $f(x_1, \ldots, x_n) \equiv -\varepsilon$.
- For this new function f, we have $\max f = -\varepsilon$. Hence, due to the inequality (5), we get
$$C - \max f \geq \frac{D + \varepsilon}{2}$$
thence (due to $\varepsilon > D$) $C - \max f > D$.
- The hypothetic algorithm U is ε-y-robustly applicable to every function f for which $C - \max f > D$, in particular, to the above constant function f. By definition of robustness, this means that U should be applicable to any function $\widetilde{f}$ which is ε-close to f, in particular, to the constant function $\widetilde{f} \equiv 0$.

The contradiction is proven, hence the hypothetic algorithm U is indeed impossible.

The theorem is proven.

9.4 Proof of Theorem 2

This proof is similar to the proof of Theorem 1:

- When $\delta < d$, then we can compute $\widetilde{d} = d - \delta > 0$. Then, whenever the global maxima of the function f are d-separated, and a function $\widetilde{f}$ is δ-x-close to f, the global maxima of $\widetilde{f}$ are $\widetilde{d}$-separated. So, as the desired robust algorithm, we can take the known algorithm corresponding to the separation $\widetilde{d} > 0$.
- When $\delta > d$, then an arbitrary function with m global maxima is δ-close to some d-separated function. Thus, if there existed such a robust algorithm we would have an algorithm which would be applicable to every function with exactly m global maxima. We have already mentioned (in the previous section) that such an algorithm is impossible.

9.5 Proof of Theorem 2′

Theorem 2′ follows from Theorem 2 if we take into consideration that the problems of solving a system of equation and of locating global maxima can be naturally (and computably) reduced to each other in such a way that the solutions to the system of equations become global maxima and vice versa (and thus, the *number* of solutions becomes the *number* of global maxima and vice versa):

- If we know how to solve systems of equations, then the problem of locating global maxima of a function $f(x_1,\ldots,x_n)$ can be reformulated as a problem of finding all solutions to an equation $f_1(x_1,\ldots,x_n)=0$, where

$$f_1(x_1,\ldots,x_n) \stackrel{\text{def}}{=} \max f - f(x_1,\ldots,x_n).$$

- Vice versa, if we know how to locate global maxima, then the problem of solving a system of equations $f_1(x_1,\ldots,x_n)=0$, $\ldots$, $f_k(x_1,\ldots,x_n)=0$ can be reformulated as a problem of finding all global maxima of a function

$$f(x_1,\ldots,x_n) \stackrel{\text{def}}{=} -(|f_1(x_1,\ldots,x_n)|+\ldots+|f_k(x_1,\ldots,x_n)|).$$

Acknowledgments

This work was partially supported by NASA under cooperative agreement NCC5-209, by the United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), and by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518.

References

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

[2] R. E. Bellman and L. A. Zadeh, "Decision-making in a fuzzy environment," Management Sci., 1970, Vol. 17, pp. B141–B164.

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

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

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

[6] R. B. Kearfott, *Rigorous global search: continuous problems*, Kluwer, Dordrecht, 1996.

[7] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Upper Saddle River, NJ, 1995.

[8] U. Kohlenbach. *Theorie der Majorisierbaben* ..., Ph.D. Dissertation, Frankfurt am Main, 1990.

[9] U. Kohlenbach, "Effective moduli from ieffective uniqueness proofs. An unwinding of de La Vallée Poussin's proof for Chebycheff approximation", *Annals for Pure and Applied Logic*, 1993, Vol. 64, No. 1, pp. 27–94.

[10] V. Kreinovich, "What does the law of the excluded middle follow from?," *Proceedings of the Leningrad Mathematical Institute of the Academy of Sciences*, 1974, Vol. 40, pp.37-40 (in Russian), English translation: *Journal of Soviet Mathematics*, 1977, Vol. 8, No. 1, pp. 266–271.

[11] V. Kreinovich, *Complexity measures: computability and applications*, Master Thesis, Leningrad University, Department of Mathematics, Division of Mathematical Logic and Constructive Mathematics, 1974 (in Russian).

[12] V. Kreinovich, "Uniqueness implies algorithmic computability", *Proceedings of the 4th Student Mathematical Conference*, Leningrad University, Leningrad, 1975, pp. 19–21 (in Russian).

[13] V. Kreinovich, Reviewer's remarks in a review of D. S. Bridges, *Constrictive functional analysis*, Pitman, London, 1979; Zentralblatt für Mathematik, 1979, Vol. 401, pp. 22–24.

[14] V. Kreinovich, *Categories of space-time models*, Ph.D. dissertation, Novosibirsk, Soviet Academy of Sciences, Siberian Branch, Institute of Mathematics, 1979, (in Russian).

[15] V. Kreinovich, "Unsolvability of several algorithmically solvable analytical problems", *Abstracts Amer. Math. Soc.*, 1980, Vol. 1, No. 1, p. 174.

[16] V. Ya. Kreinovich, *Philosophy of Optimism: Notes on the possibility of using algorithm theory when describing historical processes*, Leningrad Center for New Information Technology "Informatika", Technical Report, Leningrad, 1989 (in Russian).

[17] V. Kreinovich and R. B. Kearfott, "Computational complexity of optimization and nonlinear equations with interval data", *Abstracts of the Sixteenth Symposium on Mathematical Programming with Data Perturbation*, The George Washington University, Washington, D.C., 26–27 May 1994.

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

[19] G. W. Walster, "The Future of Intervals", *Abstracts of the 9th GAMM – IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*, Karlsruhe, Germany, September 19–22, 2000, p. 23 (full paper will appear in the conference proceedings).