Computational Complexity of Optimization and Crude Range Testing: A New Approach Motivated by 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, El Paso, TX 79968, USA
vladik@cs.utep.edu

Abstract

It is often important to test whether the maximum $\max_B f$ of a given function f on a given set B is smaller than a given number C. This "crude range testing" (CRT) problem is one of the most important problems in the practical application of interval analysis. Empirical evidence shows that the larger the difference $C - \max_B f$, the easier the test. In general, the fewer global maxima, the easier the test; and finally, the further away global maxima are from each other, the easier the test. Using standard complexity theory to explain these empirical observations fails because the compared CRT problems are all NP-hard. In this paper the analysis of fuzzy optimization is used to formalize the relative complexity of different CRT problems. This new CRT-specific relative complexity provides a new and "robust" theoretical explanation for the above empirical observations. The explanation is robust because CRT relative complexity takes numerical inaccuracy into consideration. The new explanation is important because it is a more reliable guide than empirical observations to developers of new solutions to the CRT problem.

1 Introduction

1.1 Many practical problems are optimization problems

In many real-life situations, it is important to find the best decision, control, or design. Often the best decision, control, or design is subject to given constraints. Well defined constraints are said to be *crisp*, e.g., a certain quantity q must be between 0 and 1. In other situations, the constraints are *fuzzy*, e.g., a certain quantity q should be *small*. In such situations, all the values x that satisfy fuzzy constraints form a *fuzzy set* $\mathcal{B}$; for every x, we can estimate the "degree" to which x satisfies the corresponding constraints by a real number $\mu_{\mathcal{B}}(x)$ from the interval $[0, 1]$; 1 means that we are absolutely sure that x satisfies the constraints, 0 means that we are absolutely sure that it does not, intermediate values describe different levels of uncertainty. The function which maps x into this degree is called the *membership function* of the fuzzy set. The corresponding problem of finding the best decision, control, or design is naturally formalized as a *fuzzy optimization* problem:

- find the values $x = (x_1, \ldots, x_n)$ for which the given real-valued function $f(x_1, \ldots, x_n)$ attains the largest value on the given fuzzy set $\mathcal{B}$.

When constraints are crisp, the problem of finding the best decision, control, or design is naturally formalized as a constrained optimization problem:

- given a real-valued objective function $f(x_1, \ldots, x_n)$ of several variables;
- given constraints that define the set B, the *feasible region* of all the values x that satisfy them;
- then find the values $x = (x_1, \ldots, x_n)$ for which $f(x_1, \ldots, x_n)$ attains the largest (or the smallest) value on the constraint set B.

Without loss of generality all optimization problems can be formulated either as maximizing or minimizing an objective function f by simply changing the sign of f. To simplify the exposition, except where explicitly noted, the following optimization development is framed in terms of maximizing an objective function.

1.2 Solving optimization problems requires sophisticated methods

Because optimization problems are often of practical importance, people have solved them since ancient times. If there are only a small number of possible values x, then one can simply check them all to find the best (i.e., the one for which the objective function attains its largest value). Often, however, these problems are not easy to solve, even when the constraints are crisp. In most real-life problems, the number of possible alternatives is so large that it is impossible to use an exhaustive search. To solve difficult optimization problems, ingenious algorithms are required that don't use an exhaustive search.

The practical importance of optimization problems has motivated many such algorithms to be developed. For example, one of the main reasons for inventing and developing calculus was the discovery that the maxima of a smooth function $f(x)$ on a set B are located on the border of this set and at the points x for which all partial derivatives of f are zero.

New methods of solving optimization problems are constantly being developed and old ones, improved. In particular, as discussed later in more detail, one of the most important techniques used to solve optimization problems depends on computing with intervals.

Progress solving crisp optimization problems helps to solve fuzzy optimization problems. Indeed, after the pioneering work [2] of R. Bellman and L. Zadeh – who formulated the notion of a fuzzy optimization problem – most researchers formalize fuzzy problems as corresponding crisp optimization problems. The corresponding crisp optimization problem has an objective function that combines the original objective function $f(x)$ and the membership function $\mu_{\mathcal{B}}(x)$ of the fuzzy constraint set $\mathcal{B}$.

Forming and computing the new objective function is not difficult. The difficult part is solving the resulting crisp optimization problem. Thus, any success developing more efficient crisp optimization algorithms automatically leads directly to more efficient fuzzy optimization methods.

1.3 Experience solving optimization problems can guide the development of new algorithms

When developing new optimization methods, researchers can benefit from experience applying known optimization algorithms to different problems. Before explaining how experience in the form of empirical evidence is used as an algorithm development guide, consider the following example: It is known that with more global maxima – i.e., points in a function's domain were it attains its maximum value – the more difficult it is to solve the optimization problem.

This experience is the most convincing for optimization techniques that are variants of a gradient search, which start at an arbitrary point and move in the direction of the steepest ascent. Gradient methods tend to work reasonably well when an objective function has a single maximum. However, these methods sometimes fail to work well when an objective function has several global maxima. Indeed, in this case, if large steps are taken, the algorithm may move from the attraction area of one maximum to the attraction area of another, thereby confusing the process. Smaller steps can avoid this problem, but will increase the number of iterations and drastically increase computation time. More global maxima clearly increase the difficulty of locating *all* the global maxima of an objective function.

In general, empirical evidence about problem difficulty comes from experience solving problems using known tools. With existing tools, some problems are easy to solve and some are more difficult. It is therefore natural to conclude that problem difficulty is correlated with difficulty solving them using known tools.

Algorithm developers can use this empirical evidence as a guide to select techniques and to select benchmark tests for these techniques. Specifically, it is reasonable to select:

- techniques that lead to improved performance when added to the known tools; and
- problems as benchmarks that are observed to be more difficult because these are the problems for which improved performance will be the most beneficial.

Often, this natural guidance works well, but not always.

1.4 Empirical evidence can be misleading

The trouble with empirical evidence is that it can be misleading. Breakthroughs are good examples. Breakthroughs happen when a new approach succeeds that is inconsistent with existing empirical evidence and even possibly theory.

Linear programming (see, e.g., [34]) is a good example. In linear programming, a linear function $c_1 \cdot x_1 + \ldots + c_n \cdot x_n$ is maximized over the area described by linear constraints $a_{i1} \cdot x_1 + \ldots + a_{in} \cdot x_n \leq b_i$, $1 \leq i \leq m$. It is known that, crudely speaking, in the optimal solution, n out of m inequalities must be equalities.[1] It therefore seemed natural to develop iterative methods of solving this problem in which, at every iteration, the vector $\mathbf{x}$ exactly satisfies n out of m inequalities. This method – called the *simplex-method* – turned out to be extremely empirically successful.

The simplex method is not perfect, but based on the available empirical evidence, most of the efforts aimed at improving it were restricted to techniques in which at every stage, $\mathbf{x}$ transforms n inequalities into equalities. All attempts to weaken this restriction only led to a worse algorithm. Then suddenly, completely new methods were discovered: methods that are, in many cases, much faster than the simplex method; methods in which, during each iteration, none of the inequalities are turned into equalities (see, e.g., [8, 10, 34, 36]). The available empirical evidence was misleading. If researchers had realized this fact, they might have developed the new faster methods much sooner.

[1] The proof of this fact is rather simple: If fewer than n inequalities are equalities at a given point $\mathbf{x} = x_1, \ldots, x_n$, then the n variables can be modified in such a way that n equalities remain true, and the value of the objective function is increased. Thus, the given point cannot be the maximum.

1.5 To avoid mistakes, theoretically test the corresponding empirically based hypotheses

Empirical evidence is sometimes misleading because it is based on the experience from applying known tools. An apparently difficult problem may actually be reasonably simple to solve – using unknown tools.

Since empirical evidence can be misleading, it is important to develop theoretical analyses of empirically-derived hypotheses to separate possibly misleading evidence from the evidence that is theoretically justified.

1.6 The desired theoretical analysis can be difficult

Often, the desired theoretical analysis of empirical hypothesis is difficult. There are two reasons for this.

The first is very familiar to people in the fuzzy methods community: these hypotheses are often formulated in words from natural language that are mathematically imprecise. For example, a hypothesis may state that problems from one class are "more complex" than the problems from some other class, without specifying what "more complex" means. To test such a hypothesis, it must be precisely formalized.

The second reason is that even when precisely formalized in mathematical terms, determining whether a formal hypothesis is true or not may be a complex mathematical task.

1.7 The plan

After advice to readers from the fuzzy and interval communities, two empirically based hypotheses about optimization problems are presented in Section 2. These hypotheses are believed by many researchers in the optimization community to be important, but until now have not been precisely formulated. The reason these hypotheses are important is explained in Section 3. Section 4 explains why traditional methods used to precisely formalize similar hypotheses do not work in this case.

After describing the crude range test (CRT) problem in detail, the solution is developed. This solution is based on the fact that some computational optimization problems stem from real-life problems that are naturally formalized using fuzzy optimization. As shown in Section 5, fuzzy optimization naturally provides the additional freedom needed to precisely formalize the two hypotheses of interest. Section 6 demonstrates that similar ideas make sense even for non-fuzzy practical problems. In Sections 7 and 8, the resulting precisely formalized hypotheses are described, and mathematical results are presented that confirm the two hypotheses. Proofs of these results are presented in Section 9.

1.8 Advice to readers

Because this special issue is devoted to the relation between fuzzy systems and interval analysis, the authors intend this paper to be useful for readers whose interest is either fuzzy systems or interval analysis.

1.8.1 Advice to fuzzy systems readers

For readers whose primary interest is *fuzzy systems*, the main mathematical results of this paper concern the computational complexity of crisp optimization. These results apply to fuzzy optimization only indirectly, via the fact that the standard Zadeh-Bellman formulation of fuzzy optimization problems reduces them to crisp optimization problems with a different objective function. After new definitions are motivated using fuzzy optimization, the remaining developments are mathematical, and as such, of limited interest to readers who are primarily interested in fuzzy optimization.

Nevertheless, new results may be interesting to the general fuzzy systems community because fuzzy optimization serves as the *motivation* for the precise formulation of the required crisp complexity problem. It is unfortunate that fuzzy systems concepts have not been directly applied more frequently to crisp (non-fuzzy) numerical methods. A few cases of direct applications are surveyed, e.g., in [14, 24, 28].

The present application of fuzzy methodology to the (foundations of) non-fuzzy numerical methods is new. In this case fuzzy systems concepts are applied to the development process of crisp (non-fuzzy) numerical methods. With better mutual awareness of fuzzy and non-fuzzy developments by both groups of researchers, the authors hope similar applications will become more commonplace and benefit both communities.

1.8.2 Advice to interval analysis readers

Readers whose primary interest is *interval analysis* can skip the fuzzy optimization sections in their first reading, and read only about: the problem of precisely formulating empirical hypotheses; the related optimization problem; the solution to this problem; and the resulting theorems. Nevertheless, the authors hope that interval readers are interested in the motivation for the new definitions and as a result, will read the fuzzy optimization sections.

1.8.3 General comment

The paper is aimed at both fuzzy systems and interval analysis readers. The authors want readers from both disciplines to understand and appreciate the results without having to read other textbooks or survey papers. Consequently, introductory sections contain somewhat more tutorial details than is typical in a journal article. Readers familiar with definitions and elementary ideas are welcome to skip these details.

2 Two Important Empirical Hypotheses About Optimization

In this section, two empirically based hypotheses (observations) about optimization problems are presented. Many researchers from the optimization community believe these hypotheses to be important (see, e.g., [9]). However, until now they have not been precisely formalized.

2.1 First Observation: The Closer The Maxima, The More Difficult the Problem

The first observation is easy to describe. The following observation is mentioned in the introduction:

Observation. *The problem of locating global maxima is easier if there is a single global maximum and more difficult if there are several global maxima.*

This empirical fact actually has a theoretical explanation that will be described, in some detail, in Section 8.

Until now, a second related empirical observation, however, has has not been precisely formalized:

Observation. *Locating global maxima is easier if they are widely separated and more difficult if they are close together.*

Hypothesis 1. *The closer the global maxima, the more complex the corresponding optimization problem.*

A similar observation holds for the solution to systems of nonlinear equations:

Observation. *Solving a system of nonlinear equations is easier if this system has a single solution and more difficult if the system has several solutions.*

This empirical fact actually has a theoretical explanation that will also be described, in some detail, in Section 8.

Hypothesis 1′. *The closer the solutions to nonlinear systems, the more complex the corresponding problem.*

2.2 Second Observation: Relative Complexity of Crude Range Estimation

An optimization problem consists of finding the maximum $\max\limits_B f$ of a given objective function f over a given set B. As mentioned earlier, in general, this problem is computationally difficult.

In important situations, however, knowing the *exact* maximum is not required. Instead, it is sufficient to know whether the (unknown) maximum $\max\limits_B f$ of the objective function f over a given set B is smaller than a given number C. In other words, determining the exact *range* $\left[\min\limits_B f, \max\limits_B f\right]$ (or equivalently, $\left[-\max\limits_B (-f), \max\limits_B f\right]$) of the function f on the set B is not required. Instead, it is sufficient to perform a *crude range test* (CRT) to determine whether the range of f over B is strictly less than the given value C or not.

Empirical evidence shows that different CRT problems have different relative complexity:

Hypothesis 2. *The larger the difference $C - \max\limits_B f$, the easier the problem.*

Until now, this observation has not been precisely formalized or justified.

3 Why These Hypotheses Are Important

3.1 The first problem

There seems to be no doubt about the importance of the first hypothesis, because this hypothesis is directly related to optimization, and optimization is important.

3.2 Crude range tests (CRTs)

The importance of the second hypothesis is not as clear. Indeed, from the practical viewpoint of optimization problems the maximum of the the objective function is required, together with its location. On the surface it is difficult to see a meaningful real-life problem that can be naturally formalized into a CRT.

However, CRTs are important because they a critical part of almost every interval algorithm, including those used to solve optimization problems. Solving CRTs accounts for the major part of the runtime of most interval algorithms. The reason is that the flow of control in any interval algorithm with branches is determined by the results of CRTs.

Therefore, it is no accident that in the keynote talk of the recent biannual international conference on interval computations and validated numerics [35], efficiently performing simple CRTs was mentioned as the most important problem that is currently preventing the application of interval analysis from reaching its full potential. If the relative complexity of different CRTs can be precisely formalized, reliable guidance to researchers will exist regarding which CRTs are simple and which are complex. Researchers can then focus their attention on relatively simple CRTs that are nevertheless difficult to solve using existing tools.

To describe, in detail, why CRT problems are important for optimization, first the importance of verified optimization is described. This is followed by a simple example of an interval-based verified optimization algorithm that uses CRTs. Finally, the fact that more sophisticated verified optimization techniques and most other interval algorithms also use CRTs is briefly mentioned.

3.3 Why verified optimization is often required

Many numerical algorithms for solving optimization problems end up in a *local* maximum instead of the desired global one. For example, the above-mentioned gradient method stops whenever it reaches a point where the gradient is 0 – sometimes, in a local maximum point.

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

  Consider the following two examples that are naturally minimizing optimization problems:

  - In *chemical engineering*, global minima of an energy function often describe the stable states of a system. If a global minimum is missed, a chemical reaction may go into an unexpected state, with possible serious consequences.
  - In *bioinformatics*, the actual shape of a protein corresponds to the global minimum of an energy function. If a local instead of a global minimum is found, the wrong protein geometry can result. The wrong geometry in a computer simulation testing medical uses of chemicals can cause potentially beneficial medical recommendations to be missed.

For such applications, it is critical to use *rigorous*, *automatically verified* methods of global optimization, i.e., methods that never discard an actual global maximum. For a survey of such methods, see, e.g., [7, 9].

3.4 The essence of interval-based validated optimization methods

3.4.1 The basic idea

Maximizing a function $f(x)$ over a given set B is the same thing as finding the points $x_{\rm opt}$ at which the maximum of f is attained, i.e., at which

$$f(x_{\rm opt}) = \max_B f(x).$$

The fundamental idea behind interval-based validated methods of solving optimization problems is the following:

> *If* the maximum $\max_{B'} f(x)$ of the function $f(x)$ over a subset $B' \subseteq B$ is less than the global maximum $M \stackrel{\rm def}{=} \max_B f(x)$,
>
> *then* for every $x \in B'$, we have
>
> $$f(x) < M,$$
>
> *hence* the maximum cannot be attained at any point x from set B'. Thus, all the points from B' can be deleted from the set of points where the maximum can be attained.

So, the maximum over different subsets can be used to delete the entire subsets as possible locations of the global maxima without having to perform an exhaustive search. Eventually, by eliminating large parts of the original set B, the set of possible locations of global maxima can be reduced from the original (often large) set B to a small neighborhood of the actual global maximum $x_{\rm opt}$.

The problem appears to be circular, in practice, because for the above process to work, both the global maximum of f over B and B' are required. Neither is known in practice. However, with a *lower bound* on the global maximum of f over B and an *upper bound* on the maximum of f over B', a useful algorithm can be constructed.

3.4.2 Bounds transform the basic idea to an algorithm

Because is difficult to compute the exact maximum of a function $f(x)$ over a given set, in practice neither M, the exact value of f over B, nor the exact maximum ($M' \stackrel{\rm def}{=} \max_{B'} f(x)$) over the set B' is available. However, with a lower bound $\widetilde{m}$ on M, and an upper bound $\widetilde{M'}$ on M', progress can be made. Instead of comparing the exact value M' with the global maximum M, $\widetilde{M'}$ is compared with $\widetilde{m}$. Because $M' \le \widetilde{M'}$ and $\widetilde{m} \le M$, if $\widetilde{M'} < \widetilde{m}$ it follows that $M' < M$. This conclusion is only possible with a 100% guarantee that $M' \le \widetilde{M'}$ and $\widetilde{m} \le M$. Thus, for an algorithm the requirements are for a lower bound $\widetilde{m}$ on

$\max\limits_B f(x)$ and for an upper bound $\widetilde{M}'$ on $\max\limits_{B'} f(x)$. To implement the above idea it remains necessary to compute the required bounds on the range of f over the sets B and B'.

3.4.3 Interval computations: a tool for computing range enclosures

To use the above idea, function range bounds must be computed. Arithmetic on intervals is a tool for computing bounds on the range of functions over intervals.

When the set B' is simple, e.g., when it is a box $B' = \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, with $\mathbf{x}_i = [x_i^-, x_i^+]$, and when the function $f(x)$ is one of the basic arithmetic operations – e.g., it is an arithmetic operation $f(x_1, x_2) = x_1 + x_2$, $x_1 - x_2$, $x_1 \cdot x_2$, etc. – the range of the function $f(x)$ can be explicitly computed. For example, if $f(x_1, x_2) = x_1 + x_2$, then its range $\mathbf{x}_1 + \mathbf{x}_2$ is equal to $[x_1^- + x_2^-, x_1^+ + x_2^+]$. The range of the function $f(x_1, x_2) = x_1 - x_2$ is equal to $\mathbf{x}_1 - \mathbf{x}_2 = [x_1^- - x_2^+, x_1^+ - x_2^-]$. For $f(x_1, x_2) = x_1 \cdot x_2$, the range $\mathbf{x}_1 \cdot \mathbf{x}_2$ is equal to:

$$[\min(x_1^- \cdot x_2^-, x_1^- \cdot x_2^+, x_1^+ \cdot x_2^-, x_1^+ \cdot x_2^+), \max(x_1^- \cdot x_2^-, x_1^- \cdot x_2^+, x_1^+ \cdot x_2^-, x_1^+ \cdot x_2^+)].$$

These explicit formulas are used to evaluate arithmetic expressions using interval arithmetic. To find an enclosure of a function $f(x_1, \ldots, x_n)$ on a given box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, do the following:

- first, *parse* the expression $f(x_1, \ldots, x_n)$, i.e., represent computing f as a sequence of basic arithmetic operations;
- then, replace each operation with the corresponding interval operation, and perform these operations in the original order.

For example, if $f(x) = x \cdot (1 - x)$, represent f as a sequence of two elementary operations:

- $r := 1 - x$ (r denotes the 1st intermediate result);
- $y := x \cdot r$.

In the interval version, perform the following computations:

- $\mathbf{r} := 1 - \mathbf{x}$;
- $\mathbf{y} := \mathbf{x} \cdot \mathbf{r}$.

In particular, when $\mathbf{x} = [0, 1]$, compute the intervals $\mathbf{r} := [1, 1] - [0, 1] = [0, 1]$, and

$$\mathbf{y} := [0, 1] \cdot [0, 1] = [\min(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1), \max(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1)] = [0, 1].$$

Interval arithmetic has the property that the interval obtained any algebraically equivalent interval algorithm is guaranteed to return an interval bound on the range of the function over the argument interval or box. For example, in the above case, the interval $[0,1]$ is indeed an enclosure for the actual range $[0, 0.25]$.

Sometimes it is possible by rearranging expressions to obtain narrower interval bounds. For example, in the above example, if the square of the quadratic equation is completed to yield:

$$f(x) = \frac{1}{4} - \left(x - \frac{1}{2}\right)^2,$$

then the exact range is returned if this expression is computed using interval arithmetic. Of course, an intrinsic function must be available to compute the square of an interval variable. This and interval versions of other standard intrinsic functions are available in Fortran and C++ compiler support for interval data types; see, e.g., [32, 33].

Thus, over any box, B', bounds, in particular an upper bound, on the value of f can be computed by simply evaluating f using interval arithmetic. With compiler support for interval data types, this is in principle no more difficult than computing the same expression using real arithmetic.

3.4.4 Last detail: Computing the greatest lower bound on the global maximum

The above raw idea is almost ready to implement. Only one small detail is remains: Consideration has been given to the fact that an upper bound $\widetilde{M'}$ on the value $M' = \max_{B'} f(x)$ can be computed computed using interval arithmetic. However, so far no consideration no consideration has been given to computing a lower bound $\widetilde{m}$ on the global maximum $M = \max_{B} f(x)$.

Interval optimization algorithms use a point search algorithm to find a point $\widetilde{x}_{\text{opt}} \in B$ that is believed to be close to the global maximum of f. How can the interval evaluation of f produce a lower bound m' on f that is close to m, the exact lower bound of f over B? All interval computations do is to produce the enclosure $\left[\widetilde{m'}, \widetilde{M'}\right]$ for the actual range $[m', M']$ over a box B'.

However, if $B' = \{x_{\text{opt}}\}$, the singleton set consisting of the single point (or one of the actual points) at which f attains its global maximum, then $m' = M$. If the point x_{opt} were known, a lower bound on M could be computed by simply evaluating $f(x_{\text{opt}})$ using interval arithmetic. Denote the result of this interval evaluation by $\left[f(x_{\text{opt}})^-, f(x_{\text{opt}})^+\right]$. By evaluating f at any point $\widetilde{x}_{\text{opt}} \in B$ that is in the neighborhood of x_{opt}, a valid approximate lower bound $f(\widetilde{x}_{\text{opt}})^-$ on m can be computed. By searching for values of $\widetilde{x}_{\text{opt}}$ with larger and larger approximate lower bounds, the algorithm does exactly what a normal point search algorithm does. However, the real "interval magic" results from the process of deleting sub-boxes B' for which $\widetilde{M'} < f(\widetilde{x}_{\text{opt}})^-$. The "magic" is that no sub-box B' can be deleted unless it is 100% guaranteed that every value of f

in B' is strictly less than the global maximum, M. Therefore, at the termination of the interval global optimization algorithm, only small boxes remain that are guaranteed to contain the set of all global maxima, whether there is just one, a set of them or a continuous region of values at which f attains its maximum value in B.

3.4.5 Final algorithm: simple version

Many details needed to construct an efficient interval global optimization algorithm exist. To provide a feel for the algorithm, a simple subdivision process can be used with only the most rudimentary search process for a good value of $\widetilde{x}_{\mathrm{opt}}$.The box B is subdivided into several sub-boxes B_i, and interval arithmetic is used to compute the bounds $\left[\widetilde{m}_j, \widetilde{M}_j\right]$ on the range of $f(x)$ over each box B_j. Each of the values $\widetilde{M}_j$ is an upper bound on f over the box B_j. To get a value $\widetilde{x}_{\mathrm{opt}}$, the center (or midpoint) $x_j = \mathrm{mid}\,(B_j)$ of each box can be used. When any new box B_j is processed, if $f\left(\widetilde{x}_{\mathrm{opt}}\right)^- < f\left(x_j\right)^-$, then x_j is closer to x_{opt} than $\widetilde{x}_{\mathrm{opt}}$. Therefore replace the value of $\widetilde{x}_{\mathrm{opt}}$ by the value of x_j. In this way larger lower bounds on M are produced as the algorithm proceeds. This enables more sub-boxes B_j to be deleted using a CRT.

After all possible sub-boxes have been deleted, remaining sub-boxes are subdivided and the process is repeated until sufficiently small sub-boxes remain. At any point in the algorithm, the remaining sub-boxes map the set of global solutions to the optimization problem. The sub-boxes that have been deleted are known to not contain a global maximum.

Comment. The main idea of the algorithm is as follows:

- If $C - \max\limits_{B} f$ is greater than zero then the range of f over B is less than C;
- If $\min\limits_{B} f - C$ (or equivalently, $(-C) - \max\limits_{B}(-f))$ is greater than zero, then the range of f over B is greater than C;
- If neither of the above CRT problems has an an affirmative answer then neither of the corresponding branches can be taken because the range of f over B may contain C.

When a control-flow branch cannot be resolved, the set B, which is usually an interval vector or box in n-dimensions, must be split. This is an exponential process. Thus, to drastically decrease the computation time for interval algorithms, it is necessary to efficiently solve easy CRT problems. By understanding which CRTs are relatively simple and which are relatively complex, it is possible to focus new algorithm development efforts where they have the best chance of success. That is, focus on problems that are relatively simple, but for which efficient methods do not yet exist. Understanding the relative complexity of CRTs requires that these problems be precisely formalized.

3.5 A toy example of an interval-based verified optimization algorithm that uses crude range tests (CRTs)

The above algorithm is illustrated with the following toy example: find the value of the variable x for which the function $f(x) = x \cdot (1 - x)$ attains the largest possible value on the interval $B = [0, 1]$. Of course, in this example, the solution ($x = 0.5$) is easy to obtain by simply differentiating the objective function and equating the derivative to 0. This example chosen to illustrate the basic ideas in interval global optimization on an a simple example where the computing is easy. To illustrate how rounding errors impact interval arithmetic, 3 decimal digit interval arithmetic is used.

Subdivide the original interval $B = [0, 1]$ into 10 subintervals $B_1 = [0, 0.1]$, $B_2 = [0.1, 0.2]$, $\ldots$, $B_{10} = [0.9, 1.0]$ (10 simply because it makes computations easier in this example). For each of these subintervals B_j, we use interval arithmetic to compute the corresponding enclosure $\widetilde{I}_j = \left[\widetilde{m}_j, \widetilde{M}_j\right]$ and $\left[f(x_j)^-, f(x_j)^+\right]$, where x_j is the midpoint $m(B_j)$ of the interval B_j. For example, for $B_1 = [0, 0.1]$, the intervals $\mathbf{r}_1 := [1, 1] - [0, 0.1] = [0.9, 1]$, and the enclosure is $\widetilde{I}_1 = [0, 0.1] \cdot [0.9, 1] = [0, 0.1]$. With the midpoint $x_1 = \frac{0.1}{2} = 0.05$, $\left[f(x_1)^-, f(x_1)^+\right] = [0.05, 0.05] \cdot [0.95, 0.95] = [0.0475, 0.0475]$. For B_2, $\mathbf{r}_2 = [1, 1] - [0.1, 0.2] = [0.8, 0.9]$, and the enclosure is $\widetilde{I}_2 = [0.1, 0.2] \cdot [0.8, 0.9] = [0.08, 0.18]$. With midpoint $x_2 = 0.15$, $f(x_2) = [0.15, 0.15] \cdot [0.85, 0.85] = [0.127, 0.128]$. The remaining values are contained in Table 1

j	B_j	$\left[\widetilde{m}_j, \widetilde{M}_j\right]$	x_j	$\left[f(x_j)^-, f(x_j)^+\right]$
$1, 10$	$[0, 0.1]$	$[0, 0.1]$	0.05	$[0.0475, 0.0475]$
$2, 9$	$[0.1, 0.2]$	$[0.08, 0.18]$	0.15	$[0.127, 0.128]$
$3, 8$	$[0.2, 0.3]$	$[0.14, 0.24]$	0.25	$[0.187, 0.188]$
$4, 7$	$[0.3, 0.4]$	$[0.18, 0.28]$	0.35	$[0.227, 0.228]$
$5, 6$	$[0.4, 0.5]$	$[0.2, 0.3]$	0.45	$[0.247, 0.248]$

Table 1: Toy Example Values

The greatest lower lower bound $f(x_j)^-$ at a midpoint x_j of a box B_j occurs when $j = 5$ or 6, producing the value of 0.247. Because all the upper bounds $\widetilde{M}_j$ for $j \in \{1, 2, 3, 8, 9, 10\}$ are less than this value, these sub-boxes can be deleted. Thus, the global maximum or maxima can only be located between 0.3 and 0.7.

If each of the remaining three intervals is subdivided into 10 subintervals and the above steps are repeated for the new subintervals, we conclude that

$f(x_j)^- = 0.249$, which already excludes all subintervals from the original intervals B_3, B_4, $B_{7,}$ and B_8.

3.6 More sophisticated verified optimization techniques use additional crude range tests (CRTs)

In general, validated optimization 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.

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 C of these already computed values. Thus, if we can guarantee that the maximum of the function f on a box B' is smaller than C, 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 $< C$. This checking is exactly what we called "crude range test". Thus, crude range tests are, indeed, a crucial step in solving optimization problems – and since optimization is an important practical problem, crude range tests are thus important for solving important real-life problems.

3.7 Other situations in which crude range tests (CRTs) are important

In addition to interval-based optimization, there are other situations in which crude range tests are 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 tests are an important part of these techniques.

4 Main Reason Why Formalization of the Above Empirical Hypotheses Is Difficult: Traditional Methods of Formalizing Similar Hypotheses Do Not Work Here

4.1 Traditional methods of formalizing similar hypotheses: first part

We want to formalize the statement that one general problem is more complex than some other general problem. A traditional approach to formalizing this "relative complexity" is to compare the *computational complexity* of these problems – measured by the computation time needed to solve these problems. This computational complexity can be defined as follows.

There usually exist several different algorithm for solving a general problem. For each such algorithm $\mathcal{U}$ and for each possible input x, we consider the number of elementary computational steps $t_{\mathcal{U}}(x)$ of this algorithm on this input. This number is useful because the running time of an algorithm is proportional to this number of steps. The ("worst-case") complexity $t^w_{\mathcal{U}}(n)$ of the algorithm $\mathcal{U}$ is then defined as the largest possible number of steps for all the inputs x whose size (measured, e.g., by the length of the corresponding binary string) is equal to n:

$$t^w_{\mathcal{U}}(n) \stackrel{\text{def}}{=} \max_{\text{len}(x)=n} t_{\mathcal{U}}(x).$$

The smaller $t^w(n)$, the simpler the algorithm. The complexity of a problem can be defined, crudely speaking, as the complexity of the simplest algorithm which is needed to solve the problem.

For example, if one problem can be solved by a linear-time algorithm (for which $t^w_{\mathcal{U}}(n) \sim C \cdot n$), and for another problem, it has been proven that any algorithm for solving this problem requires at least quadratic time, then the second problem is clearly more complex than the first one.

4.2 Traditional methods of formalizing similar hypotheses: second part

The above approach works well if the computational complexity is reasonable. For some problems, however, the worst-case complexity of algorithms solving this problems increase so fast that these algorithms, although theoretically possible, stop being physically feasible.

Some algorithms require lots of time to run. For some problems, all known algorithms require, for some inputs of length n, the running time proportional to $\geq 2^n$ computational steps. For reasonable sizes $n \approx 300$, the resulting running time exceeds the lifetime of the Universe and is, therefore, for all practical purposes, non-feasible.

In order to find out which algorithms are feasible and which are not, we must define, in precise terms, what "feasible" means. This definition problem has been studied in theoretical computer science; no completely satisfactory definition has yet been proposed.

The best known definition is: an algorithm $\mathcal{U}$ is *feasible* if and only if it is *polynomial time*, i.e., if and only if there exists a polynomial $P(n)$ bounding the worst-case complexity: $t^w_{\mathcal{U}}(n) \leq P(n)$ for all n.

This definition is not perfect, because there are algorithms that are polynomial time but that require billions of years to compute, and there are algorithms that require in a few cases exponential time but that are, in general, very practical. However, this is the best definition we have so far.

For many mathematical problems, it is not yet known (2001) whether they can be solved in polynomial time or not. However, it is known that some combinatorial problems are as tough as possible, in the sense that if we can solve any of these problems in polynomial time, then, crudely speaking, we can solve many practically important combinatorial problems in polynomial time. The corresponding set of important combinatorial problems is usually denoted by NP, and problems whose fast solution leads to a fast solution of all problems from the class NP are called *NP-hard*. The majority of computer scientists believe that NP-hard problems are not feasible. For that reason, NP-hard problems are also called *intractable*. For formal definitions and detailed descriptions, see, e.g., [6, 25, 26, 30].

So, if one of the problems is tractable (i.e., can be solved by a feasible algorithm), while another problem is intractable, this means that the second problem is much more complex than the first one.

Comment. The fact that a general problem is "intractable" in this sense does not necessarily mean that we cannot solve it in practice:

- First, NP-hardness means that we cannot have a *general* algorithm for solving *all* possible instance of this general problem in reasonable time. We can, however, have algorithms which solve problems from a certain *subclass*.
- Second, even if we cannot solve the problem much faster than in the exponential time 2^n, it still leaves the possibility to solve this problem for inputs of small input length n. For example, for inputs of size $n = 20$, we need $2^{20} \approx 10^6$ computational steps, which is milliseconds on any modern computer. For inputs of size $n = 30$, we need $2^{30} \approx 10^9$ steps: also quite a doable amount.

4.3 Traditional methods of formalizing similar hypotheses do not work in our case

We have already mentioned that optimization is often a very complex problem. This informal idea is confirmed by the following precise result: optimization is NP-hard (see, e.g., [23]).

Not only the optimization problem itself is NP-hard, but the crude range testing problem turns out to be NP-hard as well, even we restrict ourselves to the cases when the difference $C - \max\limits_{B} f(x)$ is large. In precise terms, the problem of computing the maximum $\max\limits_{B} f(x)$ of a given function $f(x)$ on a given box B with a given accuracy ε is NP-hard for an arbitrary ε, large or small [23].

In other words, we cannot use the traditional approach to compare the complexity of the crude range testing problems for large and for small values of the difference $C - \max\limits_{B} f(x)$, because both the problem corresponding to the large values of this difference and the problem corresponding to the small values of this difference are NP-hard.

When both compared problems are NP-hard, the traditional methodology of formalizing relative complexity does not work. We therefore need a new approach to comparing complexity of different cases of this general problem.

5 Case Study Which Helps Us Formalize (and Later Justify) the Hypotheses: Mathematical Optimization Problems Emerging from Fuzzy Optimization

5.1 Fuzzy optimization: general description

In many real-life problems, we know the exact form of the objective function $f(x)$, but the set $\mathcal{B}$ 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 real-valued function $f(x)$ over a fuzzy set $\mathcal{B}$ characterized by a membership function $\mu_{\mathcal{B}}(x)$. 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 $\mathcal{B}$ is true *and* x maximizes f. There are several ways to describe this degree in terms of $f(x)$ and $\mu_{\mathcal{B}}(x)$ (see, e.g., [11, 31]), e.g., as

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

where:

- $f_{\&}(a,b)$ is a *t-norm* (i.e., a function that estimates our degree of confidence in a composite statement $A \,\&\, B$ as $d(A \,\&\, B) \approx f_{\&}(a,b)$, where $a = d(A)$ and $b = d(B)$ are our degrees of confidence in the statements A and B);
- m and M are, correspondingly, the global minimum and the global maximum of the function $f(x)$ on, e.g., the set B of all the values x for which $\mu_{\mathcal{B}}(x) > 0$.

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

5.2 Fuzzy programming problems as the most common case of fuzzy optimization

In the above text, we formulated possible constraints in the most general form, as an arbitrary fuzzy set $\mathcal{B}$. This description is a natural analogue of the most general description of a crisp optimization problem, in which the set B of possible values of x is an arbitrary set.

In practice, the most common constraints are *inequalities* of the form $g_i(a,x) \le b_i$, where a and b are vectors, and $g(a,x)$ is a known function. For example, when the function $g(a,x)$ is linear in x, we get the above-mentioned linear programming problem. Similarly, in fuzzy optimization, most common constraints are inequalities of the type $g_i(a,x) \le b_i$, where all the components of the vectors a and b are fuzzy sets (usually, fuzzy numbers), and $g(a,x)$ is a known (real-valued) function.

By using extension principle (see, e.g., [11, 27, 29]), we can determine, for each x, the degree to which the inequality $g_i(a,x) \le b_i$ is satisfied. Using a t-norm to combine the degrees corresponding to different inequalities, we get the degree $\mu_{\mathcal{B}}(x)$ with which a given vector x satisfies all given constraints. These values form a membership function for the fuzzy constraint set $\mathcal{B}$.

5.3 Specific features of mathematical (crisp) optimization problems coming from fuzzy optimization

5.3.1 From the purely mathematical viewpoint, both crisp and fuzzy practical optimization problems are formulated as problems of crisp optimization

At first glance, 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. To be more precise, the difference is not between the resulting *mathematical* optimization problems, the difference is in the relation between the original practical problem and the resulting mathematical optimization problem:

- in the crisp case, the objective function directly reflects our preferences;
- in the fuzzy case, the objective function of the resulting crisp optimization problem is different from the function describing our preferences; specifically, this objective function is the result of combining the function describing preferences and the membership function describing fuzzy constraints.

5.3.2 In practical problems which lead to crisp optimization, the practical problem uniquely determines the resulting crisp optimization problem

In practical problems in which the constraints are crisp, the objective function $f(x)$ is precisely known, and the constraints are precisely known. These constraints can be formulated in terms of a set B of all possible alternatives x

which satisfy these constraints. By definition of the word "crisp", the resulting mathematical optimization problem is uniquely determined by the original formulation of the corresponding practical problem.

5.3.3 In contrast, the same practical fuzzy optimization problem can lead to somewhat different crisp optimization problems

A fuzzy optimization problem $f(x) \to \max\limits_{\mathcal{B}}$ is also formalized as a crisp optimization problem $\widetilde{f}(x) \to \max\limits_{B}$ – albeit with a modified objective function $\widetilde{f}(x) = \mu_M(x) = f_\& \left(\mu_{\mathcal{B}}(x), \dfrac{f(x)-m}{M-m}\right) \neq f(x)$. The difference from the case of practical crisp optimization problems is that in the fuzzy case, the same practical fuzzy optimization problem can lead to *different* crisp optimization problems.

Indeed, in practical problems which lead to fuzzy optimization, constraints are formulated by words from a natural language. For the same word like "small", different elicitation methods can lead to slightly different membership functions (see, e.g., [11]). As a result, the exact same practical constraint can lead to different membership functions $\mu_{\mathcal{B}}(x) \neq \mu'_{\mathcal{B}}(x)$.

When we substitute these different membership functions into the above expression for the the new objective function $\widetilde{f}(x) = \mu_M(x)$, we conclude that the exact same practical constraint can lead to slightly different objective functions $\widetilde{f}(x) = \mu_M(x) = f_\& \left(\mu_{\mathcal{B}}(x), \dfrac{f(x)-m}{M-m}\right)$ and $\widetilde{f'(x)} = \mu'_M(x) = f_\& \left(\mu'_{\mathcal{B}}(x), \dfrac{f(x)-m}{M-m}\right) \neq \widetilde{f}(x)$ – and thus, to slightly different crisp optimization problems.

Thus, the same real-life fuzzy optimization problem can lead not only to the objective function $\widetilde{f}(x)$, but also to other objective functions $\widetilde{f'(x)}$ which are, in some reasonable sense, close to the original function $\widetilde{f}(x)$. Thus, it makes sense to require that the algorithms not only work on a given function $f(x)$, but that they work *robustly* in the sense that they produce a correct answer not only for the exact given function $f(x)$, but for all the functions $f'(x)$ which are sufficiently "close" to this $f(x)$.

5.3.4 Close: in what sense? Simplest case of direct elicitation

Different elicitation techniques normally result in close values of the membership functions. Thus, for every x, the values $\mu_{\mathcal{B}}(x)$ and $\mu'_{\mathcal{B}}(x)$ of the membership functions obtained by using different elicitation techniques, should be close to each other. Since the values $\mu_{\mathcal{B}}(x)$ and $\mu'_{\mathcal{B}}(x)$ are close, the values of the new objective functions $\mu_M(x)$ and $\mu'_M(x)$ – which are computed correspondingly from $\mu_{\mathcal{B}}(x)$ and $\mu'_{\mathcal{B}}(x)$ – should also be close to each other.

The above argument shows, therefore, that we must consider functions $f(x)$ and $f'(x)$ "close" if, for every x, the value $f'(x)$ is close to the corresponding value of $f(x)$.

5.3.5 Close: in what sense? A more complex case of indirect elicitation

The above notion of closeness corresponds to the case when we directly obtain the values $\mu_{\mathcal{B}}(x)$ by elicitation. For example, if a constraint is that x_1 is small, a direct elicitation would mean that we ask the expert(s), for different real numbers x (e.g., for $x = 0$, $x = 0.5$, $x = 1$, etc.) to what extent this particular real number is small.

In some cases, however, the elicitation procedure is less direct. One possible reason why we may need indirect elicitation is that an expert may have difficulty explaining to what extent a given *real number* x (or, in general, a vector $x = (x_1, \ldots, x_n)$) satisfies a given property. This difficulty comes from the fact that it is often not easy to imagine a situation with a given value of x. For example, a person may have trouble answering to what extent a person is tall if his height is 1.80 m. It is much easier to say to what extent, say, President Bush is tall.

In other words, if we cannot ask an expert about the values $\mu_{\mathcal{B}}(x)$ for given x, but we can ask to what extent a given object X satisfies the given properties. In this case, we have an additional uncertainty – because we may not be 100% sure about the value of x corresponding to this test object. Instead of knowing the exact value x corresponding to this object X, we may know the interval $[x^-, x^+]$ of possible values of x. Thus, when an expert describes his or her degree μ_0 to which this object satisfies the given constraint, we can, in principle, take this value μ_0 as $\mu_{\mathcal{B}}(\bar{x})$ for different values $\bar{x}$ from this interval.

Depending on the specific elicitation procedure, we may thus represent the same expert's opinion by several different membership functions $\mu_{\mathcal{B}}(x)$ and $\mu'_{\mathcal{B}}(x)$. This difference is that for every value x, the value $\mu_{\mathcal{B}}(x)$ comes from selecting a value x from the interval $[x^-, x^+]$ corresponding to the tested object X (for which the expert marked his or her degree of constraint satisfaction as $\mu_{\mathcal{B}}(x) = \mu_0$). Another elicitation procedure may pick a different value x' from the same interval; as a result, for the corresponding membership function $\mu'_{\mathcal{B}}(x)$, we have $\mu'_{\mathcal{B}}(x') = \mu_{\mathcal{B}}(x)$ $(= \mu_0)$.

The resulting functions $\mu_{\mathcal{B}}(x)$ and $\mu'_{\mathcal{B}}(x)$ are therefore "close" in the sense that if one of these functions has a certain value at some point x, the other function should have the same (or close) value either at this same point x or at some point x' which is close to x.

5.3.6 Close: in what sense? Informal summary

In view of the above, in this paper, we will consider algorithms which are "robust" in the sense that they are applicable not only to the original function $f(x)$, but also to close functions $f'(x)$, and we will consider two types of closeness:

- first, a natural *y-closeness* which means that for every input $x = (x_1, \ldots, x_n)$, the y-values – i.e., the values of $f(x_1, \ldots, x_n)$ and $f'(x_1, \ldots, x_n)$ – are sufficiently close;
- second, an (also needed) *x-closeness*, which takes into consideration the fact that the functions $f(x)$ and $f'(x)$ may may represent the same values – but for slightly different inputs x (precise definitions are given in the following sections).

Comment. To avoid potential misunderstanding, we would like to emphasize that in this section, we are *not* proposing a new definition of a fuzzy function. All we are doing is explaining that since the same practical fuzzy optimization problem can lead to different – but close – mathematical (crisp) optimization problems, it is desirable to look for algorithms which should not change much if we replace one formalization with another formalization of the same practical problem – i.e., one objective function by another (close) one. Fuzzy optimization is used *only* as a motivation for this condition – and as a motivation for the corresponding notion of closeness (which will be defined precisely in Sections 7 and 8).

6 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 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.

7 Formalization and Justification of the Second Hypothesis: The Larger the Difference $C - \max f$, the Easier the Problem

In this and the following sections, we will describe the formalization and the justification of the above two hypotheses. For exposition purposes, it turned out to be easier to start with the second hypothesis. The first hypothesis is covered in the next section.

In order to formalize the second hypothesis, 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, 23]):

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*

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

  and

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

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

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 a crude range testing (CRT) algorithm, 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 $\max_B f < C$; and*
- *if the algorithm U returns "no", then $\max_B f \geq C$.*

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 CRT 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 CRT 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 ($\leq D$); 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:*

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

- *We say that a CRT algorithm 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 CRT algorithm which is ε-y-robustly applicable to all functions f for which $C - \max f > D$.*

- *If $\varepsilon > D$, then no CRT 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 CRT algorithm which is D-y-robustly applicable to all functions f for which $C - \max f > D_2$; and

- no CRT 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 CRT problem corresponding to D_2 is indeed easier to solve.

8 Formalization and Justification of the First Hypothesis: The Closer the Maxima, the More Difficult the Problem

8.1 Known justification of the observation that the fewer global maxima, the easier the problem

Before we describe our formalization and justification of the first hypothesis, let us recall a justification of a similar hypothesis: that the fewer global maxima, the easier the problem. This formalization and justification is described in [23], and consists of the following results:

Theorem [12, 13, 17, 19]. *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 [16, 17, 18, 19, 20, 21, 22, 23]. *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.*

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 [12, 13, 17, 19]. *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 [16, 17, 18, 19, 20, 21, 22, 23]. *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.*

8.2 Formalization and justification of the first hypothesis

In a similar manner, we can formalize th first hypothesis:

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 [12, 13, 17, 19]. *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 $\leq \delta$, i.e., for for every two points x and x', we have*

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

- *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 [12, 13, 17, 19]**.** *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.*

8.3 Can we apply these results to fuzzy optimization? A general comment to both justifications

In this paper, fuzzy optimization is used only as a motivation for the new definition of complexity. Our main complexity results are about the computational complexity of *crisp* optimization problems.

These complexity results can also be – indirectly – applied to fuzzy optimization. Indeed, from the mathematical viewpoint, many methods of fuzzy optimization can be described as crisp optimization problems – albeit with a modified objective function. Thus, e.g., from Theorem 2, we can conclude that fuzzy optimization problems which have several solutions, the closer the solutions, the more difficult the problem.

Conclusion

In many practical problems, we are looking for the best decision or the best control under given constraints. These problems are naturally formalized as optimization problems. Several efficient methods of solving optimization problems use interval computations. In applying these methods, it is often important to check whether the maximum $\max\limits_B f$ of a given function f on a given set B is smaller than a given number C.

Empirical evidence shows that different instances of this CRT problem have different relative complexity: the larger the difference $C - \max\limits_B f$, 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.

This formalization also enables us to justify another empirical fact about optimization: that in the situations when the optimized function has several global maxima, the further away global maxima from each other, the easier the problem.

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, 15, 23]). Thus, a CRT 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, 23]. 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 CRT 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
$$|M - \max f| \le 2\cdot 2^{-m}. \tag{2}$$
- Third, we use the fact that C is a computable real number and generate the rational number C_{m-1} for which
$$|C - C_{m-1}| \le 2^{-(m-1)} = 2\cdot 2^{-m}. \tag{3}$$
- 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 CRT 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 CRT 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 $C \ge C_{m-1} - 2\cdot 2^{-m}$ hence
$$C \ge C_{m-1} - 2\cdot 2^{-m} > M + 2\cdot 2^{-m}.$$

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 CRT 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) CRT 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 CRT 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 $f_1(x_1, \ldots, x_n) = 0$, ..., $f_k(x_1, \ldots, x_n) = 0$ 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 supported in part by NSF grants CDA-9522207 and 9710940 Mexico/Conacyt, by NASA under cooperative agreement NCC5-209 and grant NCC 2-1232, 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 F49620-00-1-0365, and by Grant No. W-00016 from the U.S.-Czech Science and Technology Joint Fund.

The authors are thankful to Weldon Lodwick, the editor of the special issue, for his encouragement, to Ramon E. Moore for his encouragement and useful advise, and to the anonymous referees for their very useful comments.

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] M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[7] E. Hansen, *Global Optimization Using Interval Analysis,* Marcel Dekker, 1992.

[8] N. Karmarkar, "A new polynomial-time algorithm for linear programming", *Combinatorica*, 1984, Vol. 4, pp. 373–396.

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

[10] L. G. Khachiyan, "A polynomial-time algorithm for linear programming", *Soviet Math. Dokl.*, 1979, Vol. 20, No. 1, pp. 191–194.

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

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

[13] 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.

[14] O. Kosheleva, V. Kreinovich, B. Bouchon-Meunier, and R. Mesiar, "Operations with Fuzzy Numbers Explain Heuristic Methods in Image Processing", *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'98)*, Paris, France, July 6–10, 1998, pp. 265–272.

[15] 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.

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

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

[18] 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.

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

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

[21] 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).

[22] 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.

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

[24] V. Kreinovich, H. T. Nguyen, and B. Wu, "Justification of Heuristic Methods in Data Processing Using Fuzzy Theory, with Applications to Detection of Business Cycles From Fuzzy Data", *Proceedings of the 8th IEEE International Conference on Fuzzy Systems FUZZ-IEEE'99*, Seoul, Korea, August 22–25, 1999, Vol. 2, pp. 1131–1136; extended version in *East-West Journal of Mathematics*, 1999, Vol. 1, No. 2, pp. 147–157.

[25] L. R. Lewis and C. H. Papadimitriou (1981), *Elements of the theory of computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[26] J. C. Martin, *Introduction to languages and the theory of computation*, McGraw-Hill, New York, 1991.

[27] H. T. Nguyen, "A note on the extension principle for fuzzy sets", *J. Math. Anal. and Appl.*, 1978, Vol. 64, pp. 359–380.

[28] H. T. Nguyen, V. Kreinovich, and B. Bouchon-Meunier, “Soft Computing Explains Heuristic Numerical Methods in Data Processing and in Logic Programming”, In: L. Medsker (ed.), *Frontiers in Soft Computing and Decision Systems*, AAAI Press (Publication No. FS-97-04), 1997, pp. 30–35.

[29] H. T. Nguyen and E. A. Walker, *First Course in Fuzzy Logic*, CRC Press, Boca Raton, Florida, 1999.

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

[31] R. Slowinski (ed.), *Fuzzy sets in decision analysis, operations research, and statistics*, Kluwer, Boston, Massachusetts, 1998.

[32] Sun Microsystems, *Interval arithmetic in Sun's Forte Fortran 95 compiler*, http://www.sun.com/forte/fortran/interval/index.html

[33] Sun Microsystems, *Interval arithmetic in Sun's Forte C++ compiler*, http://www.sun.com/forte/cplusplus/interval/index.html

[34] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwer, Boston, Massachusetts, 1996.

[35] 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).

[36] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, Pennsylvania, 1997.