Computational Complexity and Feasibility of Fuzzy Data Processing: Why Fuzzy Numbers, Which Fuzzy Numbers, Which Operations with Fuzzy Numbers

H.T. Nguyen
Department of
Mathematical Sciences
New Mexico State University
Las Cruces, NM 88003, USA
hunguyen@nmsu.edu

M. Koshelev
V. Kreinovich
Dept. of Computer Science
University of Texas
El Paso, TX 79968, USA
{mkosh,vladik}@cs.utep.edu

O. Kosheleva
Department of Electrical
& Computer Engineering
University of Texas
El Paso, TX 79968, USA
olga@ece.utep.edu

Abstract

In many real-life situations, we cannot directly measure or estimate the desired quantity r. In these situations, we measure or estimate other quantities $r_1, \ldots, r_n$ related to r, and then reconstruct r from the estimates for r_i. This reconstruction is called *data processing*.

Often, we only have fuzzy information about r_i. In such cases, we have *fuzzy data processing*. Fuzzy data means that instead of a single number r_i, we have several numbers that describes the fuzzy knowledge about the corresponding quantity. Since we need to process more numbers, the computation time for fuzzy data processing is often much larger than for the usual non-fuzzy one. It it, therefore, desirable to select representations and processing algorithms that minimize this increase and thus, make fuzzy data processing feasible.

In this paper, we show that the necessity to minimize computation time explains why we use fuzzy numbers, and describes what operations we should use.

1 Formulation of the problem

1.1 Why data processing

In many real-life situations, we cannot directly measure or estimate the desired quantity r. For example, we cannot directly measure the distance to a star or the amount of oil in a well.

In these situations, we measure or estimate other quantities $r_1, \ldots, r_n$ related to r, and then reconstruct r from the estimates for r_i. This reconstruction is called *data processing*.

1.2 Fuzzy data processing

In many real-life applications, we have to deal with quantities r_i whose values we do not know precisely, and instead, we only have expert (fuzzy) knowledge about these values. This knowledge is usually described in terms of *membership functions* $\mu_i(x)$ that assign to every real number x the expert's degree of belief $\mu_i(x) \in [0,1]$ that the actual (unknown) value of the quantity r_i is equal to x.

We want to use the expert (fuzzy) knowledge about the values $r_1, \ldots, r_n$ of some quantities to *predict* the value of some quantity r that is related to r_i. In this paper, we will consider the simplest case when "related" means that we know the exact form of the dependency $r = f(r_1, \ldots, r_n)$ between r_i and r, and the only uncertainty in r is caused by the uncertainty in the values of r_i.

In such situations, we must transform the fuzzy knowledge about the values r_i into a fuzzy knowledge about $r = f(r_1, \ldots, r_n)$. This transformation is called *fuzzy data processing*.

It is usually implemented by using *extension principle* (see, e.g., [12]):

$$\mu_r(x) = \sup_{x_1,\ldots,x_n : f(x_1,\ldots,x_n)=x} (\mu_1(x_1) \& \ldots \& \mu_n(x_n)), \tag{1}$$

where & is an "and"-operation (t-norm).

1.3 Fuzzy data processing takes longer than non-fuzzy one

Fuzzy data means that for each i, instead of a *single* number r_i, we have *several* numbers that describes the fuzzy knowledge about the corresponding quantity.

Since we need to process more numbers, the computation time for fuzzy data processing is often much larger than for the usual non-fuzzy one.

In some cases, formulas are still rather easy to implement: e.g., if $f(r_1, \ldots, r_n)$ is a linear function, t-norm is a product, and we use Gaussian membership functions

$$\mu_i(x) = \exp(-(x-a_i)^2/(\sigma_i)^2).$$

In this case, for $\mu_r(x)$, we also get a Gaussian membership function, with easily computable a and σ. E.g., for $f(r_1, r_2) = r_1 + r_2$, we have

$$a = \frac{a_1(\sigma_1)^{-2} + a_2(\sigma_2)^{-2}}{(\sigma_1)^{-2} + (\sigma_2)^{-2}}$$

and $(\sigma)^{-2} = (\sigma_1)^{-2} + (\sigma_1)^{-2}$ [12], [26]. These are computationally very simple formulas to implement.

There are simple formulas for several other cases (see, e.g., [12] and references therein).

However, in general, fuzzy data processing can be computationally complicated. It it, therefore, desirable to select representations and processing algorithms that minimize this increase and thus, make fuzzy data processing feasible.

1.4 Additional computational problem: defuzzification

If the result r goes to an expert, then it is better to give the expert all possible choices x with their degree of possibility $\mu_r(x)$.

However, in many real-life situations, the result of data processing goes to an automatic decision-maker (e.g., controller); in such situations, we need to *defuzzify* the membership function $\mu_r(x)$, i.e., transform it into a single number $\bar{x}$ that is, in some reasonable sense, representative of this membership function. This defuzzification requires additional computation steps. so, if we want to save computation time, we must choose as simple a defuzzification procedure as possible.

Several defuzzification procedures are known, the simplest of them is the one that goes back to the first papers of Zadeh: we choose a value $\bar{x}$ from which $\mu_r(x)$ takes the largest possible value. So, we must choose fuzzy sets and operations in such a way that at least this simplest defuzzification should be computationally feasible.

1.5 What we are planning to do

In this paper, we show that the necessity to minimize computation time explains why we use fuzzy numbers, and describes what operations we should use.

2 Why fuzzy numbers

2.1 Main idea

Even if there is no fuzzy data processing involved, there is still a need for defuzzification. So, before we consider any fuzzy data processing algorithms, we must select membership functions in such a way that will make defuzzification computationally doable. At least, the above-described simplest defuzzification must be computationally doable, in which we pick a value $\bar{x}$ for which $\mu(x)$ takes the largest possible value.

We will show that this very natural requirement implies that we should restrict ourselves to membership functions that attain maximum in exactly one point $\bar{x}$. This conclusion justifies the use of *fuzzy numbers* in fuzzy data processing.

Comment. It is a known experimental fact in numerical mathematics (see, e.g., [7], [8], [9], [10], [11]) that in general, it is *easier* to find a point $(x_1, \ldots, x_n)$, in which a given function $f(x_1, \ldots, x_n)$ attains its maximum, when there is only *one* such point, and much *harder* when there are *several.* There are several theoretical results that explain these experiments; see, e.g., [18], [19], [20], [21], [22], [23], [24], [25]. In this paper, we show that these results are true if instead of arbitrary functions, we only consider membership functions.

2.2 Definitions and known result

We are interested in computing the real number $\bar{x}$ at which a given membership function $\mu(x)$ attains its maximum. In the computer, all we have is rational numbers. What does it mean to "compute a real number"? It is natural to say that a real number is computable if we can compute its rational approximation with an arbitrary given accuracy. This definition and its analysis can be found, e.g., in [2], [4], [1], [3]):

Definition 1. *A real number x is called constructive (or 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.*

Comment.

- When we say that a constructive real number is given, we mean that we are given an algorithm that computes this real number.
- Similarly, we can define a constructive *function* from real numbers to real numbers, as a function that, given a computable real number x, computes $f(x)$ with an arbitrary accuracy.

The following result is known (see, e.g., [19], [21], [13], [14]):

Proposition. *There exists an algorithm that is applicable to an arbitrary computable function $f(x_1, \ldots, x_n)$ on a computable box $\mathbf{X} = [x_1^-, x_1^+] \times \ldots \times [x_n^-, x_n^+]$ that attains its maximum on $\mathbf{X}$ at exactly one point $x = (x_1, \ldots, x_n)$, and computes this point x.*

2.3 New result

If we allow the possibility that maximum is attained at two points instead of one, then the problem of computing this maximum becomes algorithmically decidable:

Theorem 1. *No algorithm is possible that is applicable to any computable membership function $\mu(x)$ that is different from 0 on an interval $[x^-, x^+]$ and that attains its maximum at exactly two points, and returns these two points.*

This result explains why fuzzy numbers should be used, because for a fuzzy number whose membership function is strictly increasing then strictly decreasing, the maximum is attained at exactly one point.

2.4 Proof

The proof of Theorem 1 uses the fact that it is algorithmically impossible to tell whether a real number is equal to 0 or not (see, e.g., [17], [2], [4], [1], [3]).

We will prove this Theorem by reduction to a contradiction. Assume that such an algorithm U exists. So, U is applicable to an arbitrary computable function that attains its maximum at exactly two points, and returns exactly these points. As an example of such a polynomial, let's take $g_\alpha(x) = \max(0, 1 - f_\alpha^2(x))$, where

$$f_\alpha(x) = (x - 1 - \alpha^2) \cdot (x - 1 + \alpha^2) \cdot ((x + 1)^2 + \alpha^2),$$

and α is some constructive real number.

One can easily check that $g_\alpha(x)$ is indeed a membership function (i.e., a function whose values belong to the interval $[0, 1]$), and that the only way for this function to attain the largest possible value 1 is to have $f_\alpha(x) = 0$.

It is easy to check that for every α, the polynomial $f_\alpha(x)$ has exactly two roots (i.e., points x for which $f_\alpha(x) = 0$). Indeed, $f_\alpha(x)$ is the product of three factors, so $f_\alpha(x) = 0$ if and only if one of these factors is equal to 0. We will consider two cases:

- If $\alpha = 0$, then $f_\alpha(x) = (x - 1)^2 \cdot (x + 1)^2$, so $f_\alpha(x) = 0$ if either $x = 1$, or $x = -1$.
- If $\alpha \neq 0$, then the third factor is positive, so for $f_\alpha(x)$ to be 0, one of the first two factors must be equal to 0. In other words, the roots are $x = 1 - \alpha^2$ and $x = 1 + \alpha^2$.

Now, we can get the desired contradiction: for every constructive number α, we can apply U to the polynomial $g_\alpha(x)$ and get the maxima (i.e., the roots of the polynomial $f_\alpha(x)$) with an arbitrary accuracy. Let's compute them with the accuracy 1/4. Depending on whether $\alpha = 0$ or not, we have two cases:

- If $\alpha = 0$, then one of the roots is -1, so the $(1/4)$-approximation to this root will be a negative rational number.
- If $\alpha \neq 0$, then both roots are $\geq 1 - (1/2)^2 = 3/4$, hence, their $(1/4)-$approximations are greater than 0.

So, if one of the approximations is negative, then $\alpha = 0$, else $\alpha \neq 0$. Hence, based on U, we can construct the following algorithm V that would check whether a constructive real number α is equal to 0 or not:

- apply U to $g_\alpha(x)$, and compute both roots with accuracy 1/4;
- if both resulting approximations are positive, return the answer "$\alpha \neq 0$", else return the answer "$\alpha = 0$".

But we have already mentioned that such an algorithm is impossible. So, our initial assumption (that an algorithm U exists) was wrong. The theorem is proven.

3 Which operations with fuzzy numbers

3.1 Possible choices

According to our description of fuzzy data processing, the only choice we face is the choice of selecting a t-norm ("and"-operation). There are many possible t-norms to choose from; the most well-known and the most widely used ones are the two operations contained in the original paper by Zadeh [33] that started the fuzzy logic:

- $a \& b = \min(a, b)$;
- $a \& b = a \cdot b$.

3.2 At first glance, we should choose minimum

If our only goal was only to compute $a \& b$ for two given numbers a and b, then, to minimize computation time, we should choose minimum $\min(a, b)$: indeed, on most computers, minimum is a fast hardware-supported operation (for precise formulation and proof of this conclusion, see, e.g., [27]).

3.3 In reality, minimum may not necessarily be optimal

Our actual goal is, however, more complicated: to compute the function as described by the formula (1). Of course, if we simply follow this formula, i.e., if we:

- find all possible tuples $(x_1, \ldots, x_n)$, for which $f(x_1, \ldots, x_n) = x$;
- compute $\mu_1(x_1) \& \ldots \& \mu_n(x_n)$ for each such tuple, and then
- find the largest of these values,

then the only way to minimize the computation time is to minimize the time spent on computing &, i.e., use min.

However, for fuzzy numbers, many faster methods of computing (1) are known, see, e.g., [26], [12], [6], [15], [16].

With these indirect faster methods, it may happen that minimum no longer leads to the fastest computations.

It turns out that which t-norm is the fastest depends on the function $f(x_1, \ldots, x_n)$. We will start our analysis by considering *linear* functions $f(x_1, \ldots, x_n)$, then go to quadratic ones.

3.4 What does it mean to compute a membership function?

Our goal is to compute the membership function $\mu_r(x)$. Before we start our analysis, let us re-visit the question of what it means to compute a membership function.

- *In general*, for arbitrary membership functions, it means (as we have mentioned earlier) that we are able, given x, to compute the value $\mu_r(x)$ for this x with an arbitrary accuracy.
- However, from the practical viewpoint, we are interested not so much in knowing the value $\mu_r(x)$ for a *single* given x, but rather in describing *all* values x for which the corresponding membership value exceeds a certain level α. This set (called α-cut) gives the user an indication of what values of r are possible with this possibility level. When a membership function corresponds to a fuzzy number, then for every $\alpha \in (0,1]$, the α-cut is an interval $[r^-(\alpha), r^+(\alpha)]$. From this viewpoint, to compute a membership function means to be able, for every given α, to compute the endpoints of this interval.

Comment. When we use $\min(a, b)$ as a t-norm, then we can deduce, from the extension principle (1), a simple formula for computing the desired interval:

$$[r^-(\alpha), r^+(\alpha)] =$$
$$f([r_1^-(\alpha), r_1^+(\alpha)], \ldots, [r_n^-(\alpha), r_n^+(\alpha)] =$$
$$\{f(x_1, \ldots, x_n) \,|\, x_i \in [r_i^-(\alpha), r_i^+(\alpha)]\},$$

where $[r_i^-(\alpha), r_i^+(\alpha)]$ denotes an α-cut of the membership function $\mu_i(x)$; this formula was first proposed and proven in [30] (see also [31]).

3.5 At second glance, we should also choose minimum (case of linear data processing algorithms)

Let us start with linear data processing algorithms $f(x_1, \ldots, x_n) = c_1 \cdot x_1 + \ldots + c_n \cdot x_n + c_0$.

3.5.1 Minimum

If we use *minimum* min as a t-norm, then, for this linear function, the above interval formula leads to the following expression:

$$[r^-(\alpha), r^+(\alpha)] =$$
$$c_1 \cdot [r_1^-(\alpha), r_1^+(\alpha)] + \ldots + c_n \cdot [r_n^-(\alpha), r_n^+(\alpha)] + c_0,$$

where:

- $c \cdot [x^-, x^+]$ is equal to $[c \cdot x^-, c \cdot x^+]$ if $c \ge 0$ and to $[c \cdot x^+, c \cdot x^-]$ if $c < 0$; and
- $[x_1^-, x_1^+] + \ldots + [x_n^-, x_n^+] =$ $[x_1^- + \ldots + x_n^-, x_1^+ + \ldots + x_n^+]$.

This is an easy-to-compute expression.

3.5.2 Product

If we use *product* as a t-norm, then the simplest possible computations happen when we use Gaussian membership functions, i.e., functions of the form $\exp(-P)$ for some quadratic function P (see, e.g., [26]). So, for every i, $\mu_i(x_i) = \exp(-P_i(x_i))$ for some quadratic function x_i. Let us see how for these functions, we can compute the endpoints of the desired interval for r.

According to the extension principle, $\mu_r(x) \ge \alpha$ if and only if there exist values $x_1, \ldots, x_n$ for which $f(x_1, \ldots, x_n) = x$ and $\mu_1(x_1) \cdot \ldots \cdot \mu_n(x_n) \ge \alpha$. The upper endpoint $r^+(\alpha)$ is, therefore, the largest value x for which there exist $x_1, \ldots, x_n$ such that $f(x_1, \ldots, x_n) = x$ and $\mu_1(x_1) \cdot \ldots \cdot \mu_n(x_n) \ge \alpha$. In other words, this upper endpoint is a solution to the conditional optimization problem:

$$f(x_1, \ldots, x_n) \to \max \qquad (2)$$

under the condition that

$$\mu_1(x_1) \cdot \ldots \cdot \mu_n(x_n) \ge \alpha. \qquad (3)$$

Substituting the exponential expressions for $\mu_i(x_i)$ into this inequality (3), we can reformulate its left-hand side as

$$\exp(-P_1(x_1)) \cdot \ldots \cdot \exp(-P_n(x_n)) = \exp(-z),$$

where we denoted $z = P_1(x_1) + \ldots + P_n(x_n)$. Hence, the inequality (3) is equivalent to $\exp(-z) \ge \alpha$. The function $\exp(-z)$ is strictly decreasing and therefore, this inequality is, in its turn, equivalent to $z \le A$, where we denoted $A = -\ln(\alpha)$. Thus, $r^+(\alpha)$ is a solution of the following conditional optimization problem: (2) under the condition

$$P_1(x_1) + \ldots + P_n(x_n) \le A. \qquad (4)$$

Standard methods of calculus enable us to easily solve this problem: Namely, the maximum of $f(x_1, \ldots, x_n)$ in the closed domain (4) is attained:

- either in the interior point of this domain (in which case it is a global maximum),
- or on the border of this domain.

Since a linear function does not have global maxima, the maximum must be attained on the border, i.e., when

$$P_1(x_1) + \ldots + P_n(x_n) = A. \quad (5)$$

The resulting conditional optimization problem can be then easily solved by Lagrange multiplier method, as a global maximum of a function

$$F(x_1, \ldots, x_n) =$$
$$f(x_1, \ldots, x_n) + \lambda \cdot (P_1(x_1) + \ldots + P_n(x_n)), \quad (6)$$

where the Lagrange multiplier λ can be determined from the condition (5).

The function $F(x_1, \ldots, x_n)$ is quadratic, hence, its derivatives are linear expressions and therefore, we can find its global minimum by solving the system of linear equations $\partial F/\partial x_i = 0$, $1 \le i \le n$.

Similarly, the lower endpoint $r^-(\alpha)$ of the desired interval can be obtained as a solution of a similar problem in which we minimize $f(x_1, \ldots, x_n)$ instead of maximizing it.

3.5.3 Comparison

For *linear* data processing algorithms, both the use of minimum and the use of the algebraic product lead to feasible fuzzy data processing algorithms. As can be seen from the example of addition (given above), formulas for minimum are usually slightly less complicated and require fewer computation time.

This conclusion is in good agreement with the previous conclusion about the comparative computational complexity of $\min(a, b)$ and $a \cdot b$.

Interestingly, for *non-linear* data processing algorithms, we have a reverse situation:

3.6 For quadratic data processing algorithms, we have an unexpected result: product is computationally easier than minimum

Let us now consider the simplest non-linear data processing functions $f(x_1, \ldots, x_n)$, i.e., *quadratic* ones. This time, we will start with the product.

3.6.1 Product

If we use product as a t-norm, and if we use Gaussian membership functions, then, similarly to the previous section, we can find both the lower and the upper endpoints $r^-(\alpha)$ and $r^+(\alpha)$ of the desired interval by finding the global minimum and maximum of a quadratic function (6). Similarly to the previous section, we can therefore conclude that this computation can be done by a *feasible* algorithm. Thus, we can make the following conclusion:

Theorem 2. *When we use algebraic product as a t-norm, and when all membership functions are Gaussian, then for quadratic data processing functions $f(x_1, \ldots, x_n)$, fuzzy data processing can be done by a feasible algorithm.*

3.6.2 Minimum

For minimum, the situation is quite different. The problem here is equivalent to computing the interval range of a given quadratic function $f(x_1, \ldots, x_n)$ over given intervals $[r_1^-(\alpha), r_1^+(\alpha)], \ldots, [r_n^-(\alpha), r_n^+(\alpha)]$. This problem is known to be *computationally intractable* (*NP-hard*) ([32], [25]; for a brief intro to this notion see the appendix). Thus, we can make the following conclusion:

Theorem 3. *When we use minimum as a t-norm, then for quadratic data processing functions $f(x_1, \ldots, x_n)$, fuzzy data processing is NP-hard.*

3.7 Conclusion

If we want to minimize the computation time of fuzzy data processing, then:

- If we need no data processing at all, or if we only need linear data processing, then it is better to use minimum as a t-norm.
- If we need non-linear data processing, then it is better to use the algebraic product.

	$a\&b = \min(a,b)$	$a\&b = a \cdot b$
No data processing	feasible, slightly easier *	feasible slightly more complicated
Linear data processing	feasible, slightly easier *	feasible slightly more complicated
Quadratic data processing	NP-hard	feasible *

Acknowledgments.

This work was supported in part by NASA under cooperative agreement NCCW-0089, by NSF grants No. DUE-9750858, EEC-9322370, and CDA-9522207, 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] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, 1967.

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

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

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

[6] K. Hirota and M. Sugeno, *Industrial Applications of Fuzzy Technology in the World*, World Scientific, Singapore, 1996.

[7] R. B. Kearfott, Some tests of generalized bisection, *ACM Trans. Math. Softw.*, 1987, Vol. 13, pp. 197–220.

[8] R. B. Kearfott, Interval arithmetic techniques in the computational solution of nonlinear systems of equations: Introduction, examples, and comparisons, In: *Computational solution of nonlinear systems of equations, Proc. SIAM-AMS Summer Semin., Ft. Collins/CO (USA) 1988*, American Math. Society, Providence, RI, Lectures in Appl. Math. 1990, Vol. 26, pp. 337–357.

[9] R. B. Kearfott, Interval Newton/generalized bisection when there are singularities near roots, *Ann. Oper. Res.*, 1990, Vol. 25, No. 1–4, pp. 181–196.

[10] R. B. Kearfott, Preconditioners for the interval Gauss-Seidel method, *SIAM J. Numer. Anal.*, 1990, Vol. 27, No. 3, pp. 804–822.

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

[12] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications,* Prentice Hall, Upper Saddle River, NJ, 1995.

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

[14] U. Kohlenbach, Effective moduli from effective 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.

[15] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT), *Proceedings of the 1996 IEEE International Conference on Fuzzy Systems* (New Orleans, September 8–11, 1996) Vol. 3, 1958–1964.

[16] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT), *Fuzzy Sets and Systems*, 1997, Vol. 91, No. 2, pp. 269–277.

[17] 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.

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

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

[20] 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.

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

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

[23] 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).

[24] 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.

[25] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997 (to appear).

[26] V. Kreinovich, C. Quintana, and L. Reznik, Gaussian membership functions are most adequate in representing uncertainty in measurements, *Proceedings of NAFIPS'92: North American Fuzzy Information Processing Society Conference, Puerto Vallarta, Mexico, December 15–17, 1992*, NASA Johnson Space Center, Houston, TX, 1992, pp. 618–625.

[27] V. Kreinovich and D. Tolbert, Minimizing computational complexity as a criterion for choosing fuzzy rules and neural activation functions in intelligent control, In: M. Jamshidi, C. Nguyen,

R. Lumia, and J. Yuh (Editors), *Intelligent Automation and Soft Computing. Trends in Research, Development, and Applications. Proceedings of the First World Automation Congress (WAC'94), August 14–17, 1994, Maui, Hawaii*, TSI Press, Albuquerque, NM, 1994, Vol. 1, pp. 545–550.

[28] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Inc., New Jersey, 1981.

[29] J. C. Martin, *Introduction to languages and the theory of computation*, McGraw-Hill, N.Y., 1991.

[30] H. T. Nguyen, A note on the extension principle for fuzzy sets, *J. Math. Anal. and Appl.* **64** (1978) 359–380.

[31] H. T. Nguyen and V. Kreinovich, Nested Intervals and Sets: Concepts, Relations to Fuzzy Sets, and Applications, In: R. B. Kearfott et al (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996, pp. 245–290.

[32] S. A. Vavasis, *Nonlinear optimization: complexity issues*, Oxford University Press, N.Y., 1991.

[33] L. Zadeh, Fuzzy sets, *Information and control*, 1965, Vol. 8, pp. 338–353.

4 Appendix: The notions of feasibility and NP-hardness – brief introduction

4.1 What does "feasible" mean? the main idea

Some algorithms are not feasible. In theory of computation, it is well known that not all algorithms are feasible (see, e.g., [5], [28], [29]): whether an algorithm is feasible or not depends on how many computational steps it needs.

For example, if for some input x of length $\text{len}(x) = n$, an algorithm requires 2^n computational steps, then for an input of a reasonable length $n \approx 300$, we would need 2^{300} computational steps. Even if we use a hypothetical computer for which each step takes the smallest physically possible time (the time during which light passes through the smallest known elementary particle), we would still need more computational steps than can be performed during the (approximately 20 billion years) lifetime of our Universe.

A similar estimate can be obtained for an arbitrary algorithm whose running time $t(n)$ on inputs of length n grows at least as an exponential function, i.e., for which, for some $c > 0$, $t(n) \geq \exp(c \cdot n)$ for all n. As a result, such algorithms (called *exponential-time*) are usually considered *not feasible.*

Comment. The fact that an algorithm is not feasible, does not mean that it can never be applied: it simply means that there are cases when its running time will be too large for this algorithm to be practical; for other inputs, this algorithm can be quite useful.

Some algorithms are feasible. On the other hand, if the running time grows only as a polynomial of n (i.e., if an algorithm is *polynomial-time*, then the algorithm is usually quite feasible.

Existing definition of feasibility: the main idea. As a result of the above two examples, we arrive at the following idea: An algorithm $\mathcal{U}$ is called *feasible* if and only if it is *polynomial-time*, i.e., if and only if there exists a polynomial $P(n)$ such that for every input x of length $\text{len}(x)$, the computational time $t_{\mathcal{U}}(x)$ of the algorithm $\mathcal{U}$ on the input x is bounded by $P(\text{len}(x))$: $t_{\mathcal{U}}(x) \leq P(\text{len}(x))$.

In most cases, this idea works. In most practical cases, this idea *adequately* describes our intuitive notion of feasibility: *polynomial-time* algorithms are usually *feasible*, and *non-polynomial-time* algorithms are usually *not feasible.*

This idea is not perfect, but it is the best we can do. Although in *most* cases, the above idea adequately describes the intuitive notion of feasibility, the reader should be warned that this idea is *not perfect*: in some (very rare) cases, it does not work (see, e.g., [5], [28], [29]):

- Some algorithms are polynomial-time but not feasible: e.g., if the running time of an algorithm is $10^{300} \cdot n$, this algorithm is polynomial-time, but, clearly, not feasible.
- Vice versa, there exist algorithms whose computation time grows, say, as $\exp(0.000\ldots01 \cdot \text{len}(x))$. Legally speaking, such algorithms are exponential time and thus, not feasible, but for all practical purposes, they are quite feasible.

It is therefore desirable to look for a *better* formalization of feasibility, but as of now, "polynomial-time" is the best known description of feasibility.

4.2 When is a problem tractable?

What would be an ideal solution. At first glance, now, that we have a definition of a feasible algorithm, we can describe which problems are tractable and which problems are intractable: If there exists a polynomial-time algorithm that solves all instances of a problem, this problem is tractable, otherwise, it is intractable.

Sometimes, this ideal solution is possible. In some cases, this ideal solution is possible, and we either have an explicit polynomial-time algorithm, or we have a proof that no polynomial-time algorithm is possible.

Alas, for many problems, we do not know. Unfortunately, in many cases, we do not know whether a polynomial-time algorithm exists or not. This does not mean, however, that the situation is hopeless: instead of the missing *ideal* information about intractability, we have another information that is almost as good:

What we have instead of the ideal solution. Namely, for some cases, we do not know whether the problem can be solved in polynomial time or not, but we do know that this problem is as hard as practical problems can get: if we can solve *this* problem easily, then we would have an algorithm that solves *all* problems easily, and the existence of such universal solves-everything-fast algorithm is very doubtful. We can, therefore, call such "hard" problems *intractable*. Formally, these problems are called *NP-hard.*

In order to formulate this notion in precise terms, we must describe what we mean by a problem, and what we mean by the ability to *reduce* other problems to this one.

4.3 How can we define a general practical problem?

What is a practical problem: informal idea. What is a practical problem? When we say that there is a practical problem, we usually mean that:

- we have some information (we will denote its computer representation by x), and
- we know the relationship $R(x, y)$ between the known information x and the desired object y.

In the computer, everything is represented by a binary sequence (i.e., sequence of 0's and 1's), so we will assume that x and y are binary sequences.

Two examples of problems. In this section, we will trace all the ideas on two examples, one taken from mathematics and one taken from physics. Readers who do not feel comfortable with one of the example (say, with a physical one) are free to simply skip it.

- (Example from *mathematics*) We are given a mathematical statement x. The desired object y is either a proof of x, or a "disproof" of x (i.e., a proof of "not x"). Here, $R(x, y)$ means that y is a proof either of x, or of "not x".
- (Example from *physics*) x is the results of the experiments, and the desired y is the formula that fits all these data. Imagine that we have a series of measurements of voltage and current: e.g., x consists of the following pairs $(x_1^{(k)}, x_2^{(k)})$, $1 \le k \le 10$: $(1.0, 2.0), (2.0, 4.0), \ldots, (10.0, 20.0)$; we want to find a formula that is consistent with these experiments (e.g., y is the formula $x_2 = 2 \cdot x_1$).

Solution must be checkable. For a problem to be practically meaningful, we must have a way to check whether the proposed solution is correct. In other words, we must assume that there exists a feasible algorithm that checks $R(x, y)$ (given x and y). If no such feasible algorithm exists, then there is no criterion to decide whether we achieved a solution or not.

Solution must not be too long. Another requirement for a real-life problem is that in such problems, we usually know an *upper bound* for the length $\mathrm{len}(y)$ of the description of y. In the above examples:

- In the *mathematical* problem, a proof must be not too huge, else it is impossible to check whether it is a proof or not.
- In the *physical* problem, it makes no sense to have a formula $x_2 = f(x_1, C_1, \ldots, C_{40})$ with, say, 40 parameters to describe the results $(x_1^{(1)}, x_2^{(1)}), \ldots, (x_1^{(10)}, x_2^{(10)})$ of 10 experiments.

In all cases, it is necessary for a user to be able to read the desired solution symbol-after-symbol, and the time required for that reading must be feasible. In the previous section, we have formalized "feasible time" as a time that is bounded by some polynomial of $\mathrm{len}(x)$. The reading time is proportional to the length $\mathrm{len}(y)$ of the answer y. Therefore, the fact the reading time is bounded by a polynomial of $\mathrm{len}(x)$ means that the length of the output y is also bounded by some polynomial of $\mathrm{len}(x)$, i.e., that $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$ for some polynomial P_L.

So, we arrive at the following formulation of a problem:

Definition. *By a general practical problem (or problem from the class NP), we mean a pair $\langle R, P_L \rangle$, where $R(x, y)$ is a feasible algorithm that transforms two binary sequences into a Boolean value ("true" or "false"), and P_L is a polynomial.*

Definition. *By an instance of a (general) problem $\langle R, P_L \rangle$, we mean the following problem:*

> *GIVEN: a binary sequence x.*
>
> *GENERATE either y such that $R(x, y)$ is true and $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$, or, if such a y does not exist, a message saying that there are no solutions.*

For example, for the general mathematical problem described above, an instance would be: given a statement, find its proof or disproof.

Comments. Problems for which there is a feasible algorithm that solves all instances are called *tractable*, or "problems from the class P". It is widely believed that not all problems are easily solvable (i.e., that NP$\neq$P), but it has never been proved.

One way to solve an NP problem is to check $R(x, y)$ for all binary sequences y with $\mathrm{len}(y) \le P_L(\mathrm{len}(x))$. This algorithm requires exponential time ($2^{P_L(\mathrm{len}(x))}$) and is therefore, not feasible.