How to make constrained fuzzy arithmetic efficient

Mirko Navara and Zdeněk Žabokrtský *
Center for Machine Perception
Faculty of Electrical Engineering
Czech Technical University
Technická 2, 166 27 Praha, Czech Republic
navara@cmp.felk.cvut.cz, zabokrtz@cs.felk.cvut.cz

Abstract

In the standard fuzzy arithmetic, the vagueness of fuzzy quantities always increases. G.J. Klir [2, 3] suggests an alternative—the constrained fuzzy arithmetic—which reduces this effect. On the other hand, it significantly increases the complexity of computations in comparison to the classical calculus of fuzzy quantities.

So far, little attention was paid to the problems of implementation of the constrained fuzzy arithmetic, especially to its computational efficiency. We point out the related problems and outline the ways of their solution. We suggest to decompose the whole expression, classify all its subexpressions with respect to their individual computational complexity and precompute the corresponding subresults according to this classification.

Key words: Fuzzy number, fuzzy interval, fuzzy arithmetic.

1 Introduction

Modern knowledge-based systems often require work with uncertainty information. One of the tools describing vagueness of data is fuzzy logic. It uses the real interval $[0, 1]$ as the set of truth values, as well as membership degrees of fuzzy sets. Fuzzy logical operations are used to evaluate logical formulas.

Fuzzy quantities are special types of fuzzy subsets of the real line. They generalize the classical (crisp) real numbers by their "approximate" fuzzy extensions. For calculations with them, several systems of operations were suggested. We compare here two of them, the standard fuzzy arithmetic (developed

*This research was supported by the Czech Ministry of Education under Research Programme J04/98:212300013 Decision and Control for Industry.

in [6] and described in full detail in [1, 4]) and the constrained fuzzy arithmetic suggested in [2, 3].

1.1 Basic notions

We deal here with fuzzy subsets of the real line, $\mathbb{R}$, i.e., with mappings from $\mathbb{R}$ to the unit interval $[0,1] \subseteq \mathbb{R}$. By a *fuzzy quantity* (*fuzzy interval*) we mean a mapping $A\colon \mathbb{R} \to [0,1]$ satisfying the following three conditions:

1. convexity and closedness: for each $\alpha \in (0,1]$, the *α-cut* ${}^{\alpha}A = \{x \in \mathbb{R} : A(x) \geq \alpha\}$ is a closed interval,
2. boundedness: the *support* of A, $\operatorname{Supp} A = \{x \in \mathbb{R} : A(x) > 0\}$, is bounded,
3. normality: the *core*, ${}^{1}A = \{x \in \mathbb{R} : A(x) = 1\}$, is nonempty.

If, moreover, the core of A is a singleton, we call A a *fuzzy number*. As a consequence of this definition, for each fuzzy interval A there are numbers $a, b, c, d \in \mathbb{R}$, $a \leq b \leq c \leq d$, such that

$$(a,d) \subseteq \{x \in \mathbb{R} : A(x) > 0\} \subseteq [a,d]\,,$$

$$\{x \in \mathbb{R} : A(x) = 1\} = [b,c]\,,$$

$$A \text{ is nondecreasing on } [a,b]\,,$$

$$A \text{ is nonincreasing on } [c,d]\,.$$

In particular, a *trapezoidal fuzzy interval* A is piecewise linear on the above intervals; it is fully determined by the quadruple $\langle a,b,c,d\rangle$ as follows:

$$A(x) = \begin{cases} \frac{x-a}{b-a} & \text{when } x \in [a,b)\,, \\ 1 & \text{when } x \in [b,c]\,, \\ \frac{d-x}{d-c} & \text{when } x \in (c,d]\,, \\ 0 & \text{otherwise.} \end{cases}$$

If, moreover, $b = c$, we speak of a *triangular fuzzy number* parameterized by the triple $\langle a,b,d\rangle$.

The mapping $h_A\colon (0,1] \to \exp \mathbb{R}$, defined by $h_A(\alpha) = {}^{\alpha}A$, determines A uniquely, giving the *horizontal representation* of A. (To emphasize the difference, we speak of the representation of a fuzzy set by the mapping $A\colon \mathbb{R} \to [0,1]$ as the *vertical representation*.)

The horizontal representation is advantageous for the computer implementation. While we usually have to distinguish a finite, but very large number of real values, say u, it is usually sufficient to restrict attention to a much smaller number of membership degrees α; let us denote this number by k. For a fuzzy quantity A, each α-cut is a closed interval, so its horizontal representation requires to record only $2k$ real numbers (bounds of α-cuts). For general shapes of membership functions, the vertical representation would require u real numbers (membership degrees).

1.2 Standard fuzzy arithmetic

The basic aim of the fuzzy arithmetic is to extend the operations $+, -, \cdot, /$ to fuzzy intervals. Let $\Box \in \{+, -, \cdot, /\}$. In the standard fuzzy arithmetic (SFA), the operation $\Box$ is extended to fuzzy intervals A, B in the following two equivalent ways[1]:

1. in the vertical representation by the extension principle

$$(A \Box B)(c) = \sup_{a \Box b = c} \min\{A(a), B(b)\},$$

2. in the horizontal representation

$${}^{\alpha}(A \Box B) = \{a \Box b \mid (a, b) \in {}^{\alpha}(A \times B)\}.$$

1.3 Constrained fuzzy arithmetics

Let us quote G.J. Klir who introduced the constrained fuzzy arithmetic (CFA) [2]:

> When fuzzy arithmetic is employed for dealing with fuzzy systems which are viewed as systems of linguistic variables, it is essential to take into account all information regarding the linguistic variables involved. It is argued that the standard fuzzy arithmetic does not utilize some of the information available. As a consequence, it may produce results that are more imprecise than necessary or, possibly, even incorrect.

For example, we are not satisfied with the fact that $A - A$ is not equal to crisp zero when applying the standard fuzzy arithmetics, though the linguistic variable A is connected to one real variable only. Therefore, only a pair of equal operands instead of any combination of them should be used. This limitation, called an *equality constraint*, gives the desired result also in the case of A/A. G.J. Klir wrote:

> ...the evaluation of any algebraic expression involving arithmetic operations on fuzzy intervals must take into account the equality constraint for each group of fuzzy intervals that are represented by the same symbol. The elementary fuzzy arithmetic operation $\Box$ under the equality constraint E may be expressed for all $\alpha \in (0, 1]$ by the equation

$${}^{\alpha}(A \Box A)_E = \{a \Box a \mid a \in {}^{\alpha}A\}$$

[1] Some problems may arise with division by zero. This case should be avoided. We do not deal with these questions here.

See the difference with respect to the standard fuzzy arithmetic:

$$^{\alpha}(A \square A) = \{a_1 \square a_2 \mid (a_1, a_2) \in {}^{\alpha}(A \times A)\}$$

To avoid any confusion, we should emphasize that

$$^{\alpha}(A \square B)_{\mathrm{CFA}} = \{a_1 \square a_2 \mid (a_1, a_2) \in {}^{\alpha}(A \times B)\}$$

is the same in both the constrained and unconstrained fuzzy arithmetics, even in the case when fuzzy sets A and B are absolutely equal and have only different names.

The names of variables are used to determine that the same variable can appear at several positions in the expression. (Similar situation appears in probability theory, where calculations with random variables give different results if we know that two variables are the same, hence with correlation 1.)

2 Why is CFA computationally difficult?

First, let us briefly discuss how a standard fuzzy expression could be evaluated. We prefer the α-cut method to the extension principle since this way we can concentrate only on the endpoints of intervals instead of computing the suprema on continuous domains. The fuzzy interval task is then decomposed into k crisp interval computations on α-levels which are usually uniformly distributed on the interval $[0, 1]$. The following equalities hold

$$[a, b] + [c, d] = [a + c, b + d]$$

$$[a, b] - [c, d] = [a - d, b - c]$$

$$[a, b] \cdot [c, d] = [\min(a \cdot c, a \cdot d, b \cdot c, b \cdot d), \max(a \cdot c, a \cdot d, b \cdot c, b \cdot d)]$$

$$[a, b]/[c, d] = [\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)], 0 \notin [c, d]$$

Then, the result is evaluated the same way as for real numbers—that means by decomposition of the whole expression into a sequence of binary operations, we just take care of operators' priority—for example via a pushdown automaton and LL(1) grammar. Finally, the output fuzzy set is composed from individual resulting intervals which are interpreted as its α-cuts.

The computational complexity of an interval expression is linear with respect to the number of operands n ($n = 2$ for $A + A$). The computational complexity of a fuzzy interval expression is then $\mathcal{O}(k \cdot n)$.

But now, the crucial disappointment comes. Since associativity is not generally guaranteed in CFA, the evaluation of the expression cannot be decomposed into a sequence of binary operations and the computation must be done globally.

Moreover, when evaluating endpoints only, we will obtain exact result only in the case of functions which are monotonic on the multidimensional domain composed of supports of individual operands. (For instance, when we have a trapezoidal linguistic variable A parameterized with $\langle -2, 0, 0, 2\rangle$, the support of

the expression $A \cdot A$ does not contain negative numbers, since we cannot employ, e.g., the pair -1 and 1 any more in CFA.) It is useless to urge that the intent of monotonicity cannot be accepted.

We try to formulate the problem of CFA more clearly. On every α-level, we are to find both the minimum and the maximum of a given expression—the endpoints of the resulting interval (taking these intervals one after another, we will again reconstruct the total result). The domain of our search is a multidimensional interval, in each dimension it corresponds to an α-cut of an individual linguistic variable. The total dimension equals the number of distinct operands.

Discontinuity can be avoided (it appears only when we divide by "fuzzy zero"), but steep continuous functions are obtained, e.g., as high order polynomials. Finding their extremes is a task that is not algorithmizable in general. The blind search would lead to the complexity of order $n \cdot u^n$, where n is the number of distinct variables in the formula. This situation is unsatisfactory. Therefore we have to use tools that allow to simplify the calculation at least in some cases.

3 Our approach

3.1 Decomposition

In [3] G.J. Klir writes that the computation of an expression in the constrained fuzzy arithmetic must be done globally and cannot be decomposed into a sequence of binary operations as in the standard fuzzy arithmetic. Unfortunately, this assumption forces us to make high amount of computations when evaluating such an expression. But not all of this work is always inevitable. That is why we would like to return at least to the partial decomposition in the cases when it does not spoil the result.

Under the term *decomposition* we understand splitting the primary expression into two or more new subexpressions which have mutually disjoint sets of variables.

If an expression cannot be further decomposed, we say it is *irreducible*, otherwise it is *reducible*.

The aim is to decompose the expression into subexpressions as small as possible. Instead of the global computation of the original expression, only these irreducible subexpressions must be computed "globally". This is the key point, since the computational cost of the expression does not grow linearly with the length of the expression in the constrained fuzzy arithmetic, therefore it is more advantageous to evaluate globally individual subexpressions one after another and then put their subresults together instead of the global computation of the whole original expression.

We finish the evaluation by putting the subresults together. Since the decomposition guarantees that there is no equality constraint among the subexpressions, we can use the standard fuzzy arithmetics in this step.

Example of decomposition:

$$((C \cdot A \cdot (A + B)) \cdot C + D)_{\mathrm{CFA}} = (A \cdot (A + B))_{\mathrm{CFA}} \cdot (C \cdot C)_{\mathrm{CFA}} + D$$

It is not trivial to implement an efficient general procedure for the maximum decomposition of expressions. On the other hand, at least an imperfect decomposition would be helpful as well.

We show an easy example of such an imperfect decomposition procedure. It consists of several steps:

1. After parsing (e.g., the standard LR grammar) of the string which contains the expression, generate a binary tree with leaf nodes corresponding to the operands (fuzzy variables or constants) and non-leaf nodes corresponding to the basic arithmetic operators.

2. In the node corresponding to the subtraction or the division operator, move this operator to the right subtree and change the original node to the commutative counterpart of the original operator. Example: $A - B = A + (-B)$ or $A/B = A * (1/B)$.

3. Rebuilt the binary tree into the tree where both commutative operators are viewed as n-ary, where n is as high as possible.

4. In each node corresponding to a commutative operator, make a permutation of its sequence of subtrees such that the subtrees sharing a variable are adjacent.

5. Go through the tree in the top-down direction and condense all the subtrees containing the same variable into one single node.

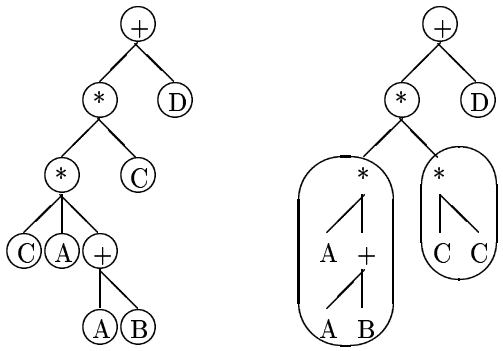

Figure 1: The effect of permutation and condensation

More advanced methods should make use also of associativity and distributivity of the operations to find possible decompositions of this type even when they are not possible in the original form.

3.2 Classification

Instead of using a general algorithm for evaluation of any constrained fuzzy expression, we first classify the expression and then we use the method which is the most efficient for the corresponding class. The following structure of the set of all constrained fuzzy expressions was observed:

- *Simple fuzzy expressions* (class S) are those which do not contain any fuzzy variable more than once, no equality constraint can occur and therefore they can be handled as in the standard (unconstrained) fuzzy arithmetic. Example: $A + B + 1$.

- *Monotonic equality-constrained fuzzy expression* (class M) is monotonic with respect to all variables on the domain given by the cartesian product of supports of all considered fuzzy variables. For each $\alpha \in (0, 1]$, the two bounds of the α-cut of the result can be computed directly.

  Example: $2 \cdot A - A + B \cdot B \cdot B$.

  Very often monotonicity is achieved only under some assumption on the variables (which can be easily checked).

  Example: $A \cdot A - B/C$, where A is nonnegative and $0 \notin \operatorname{Supp} B \cup \operatorname{Supp} C$.

  The computational complexity is the same as for standard fuzzy arithmetic—it grows linearly with the length of the expression.

- For the *vertex equality-constrained fuzzy expressions* (class V) the following holds: For each expression E in V with variables $A_1, \ldots, A_n$ and $\beta \in (0, 1]$, the bounds of the interval ${}^{\beta}E$ are of the form $E(\vec{x})$, where $\vec{x}$ is a vertex of the multidimensional interval $\prod_{i \le n} {}^{\alpha}A_i$ for some $\alpha \in [\beta, 1]$ or $\vec{x} \in \prod_{i \le n} {}^{1}A_i$.

  Example: $(p \cdot A + q \cdot B) \cdot (p \cdot A + q \cdot B)$, where $p, q \in \mathbb{R}$.

  This class still admits an effective calculation by a method which we call the *vertex algorithm* [5]. The only problem remains with the 1-cuts (=cores); here the global search has to be performed. (For fuzzy numbers, the cores are singletons and no problem arises.)

- *Equality-constrained fuzzy expressions* (class G) form the most general class satisfying no special assumptions.

Obviously, $S \subset M \subset V \subset G$. All these four classes are closed under decomposition.

3.3 Vertex algorithm

The class V is the largest class from our hierarchy where an efficient evaluation is still possible. It is broader than the class M, since for some nonmonotonic expressions the extreme values can be found on the vertices of the multidimensional intervals determined by the α-cuts.

Trying to facilitate the computation, we returned to the method of examining only the endpoints. Instead of searching in the multidimensional interval, we compare only the values in its vertices. For k (the number of α-cuts) sufficiently large, we may expect that the expression satisfying the requirements of the class V is "almost" monotonic between the borders of two successive α-cuts. If the continuous function is not monotonic over the whole domain, it is at least piecewise continuous. Increasing k, we get closer and closer to the endpoints of these pieces.

Naturally, this expectation decays in the case when there is some constant segment of the membership function of any operand within the support of this operand. This cannot happen if we take into account only trapezoidal fuzzy sets (including special cases—fuzzy numbers, real numbers and crisp intervals). Then the only regions of nonzero constant membership functions are the cores. Unfortunately, it is a very frequent case that "something happens" in the cartesian product of cores, e.g., $A \cdot A$ when A is the trapezoidal fuzzy number $\langle -2, -1, 1, 2 \rangle$; we do not want to loose this sort of CFA expressions. The conclusion is that the extensive search of extremes in the cartesian product of the cores should be performed as well.

Now, if we proceed in the top-down direction along the membership value axis (α is decreasing), we can approximate both extremes of the current multidimensional interval as the maximum or minimum of values at all its vertices and the extremes from the preceding α-cut, which is a subset of the current one. Only the multidimensional interval formed by the cores of individual fuzzy intervals should be explored in a more exhaustive way, because it has no preceding subset.

In contrast to the evaluation of expressions from the class M, this is really an approximation. When increasing k for expressions from the class M, we only get more "points" for the horizontal representation, all of them being exact. But when computing an expression from the class V which contains higher order polynomials, the resulting "points" could be inexact for finite k.

Fortunately, polynomials of small orders do not produce significant errors. On the contrary, a polynomial of high order with malicious parameters could severely endanger the correctness of the result when using small k. But the computational cost of the solution which would find their extremes reliably would be probably significantly higher. We can only believe that such polynomials usually do not occur in the area of practical fuzzy applications. The higher degree of a polynomial may occur, the higher number of α-cuts is to be used.

Let us describe the **vertex algorithm** for evaluating an expression $f(\vec{x})$ in details:

1. Let $\alpha := 1$; choose k as the number of α-cuts; $\Delta := \frac{1}{k-1}$ (the step for decreasing α)
2. By some iterative method find or estimate values of variables MAX and MIN as maximum and minimum of the given function in the domain given by cores of the fuzzy intervals which were used as operands, and save them as two ordered pairs, $\langle MIN, 1\rangle$ and $\langle MAX, 1\rangle$
3. Let $\alpha := \alpha - \Delta$
4. Construct the set $S := \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{2^n}\}$ of all vertices of the multidimensional crisp interval corresponding to the new α-cuts (n is the number of distinct variables)
5. Let
$$MAX := \max\left(\bigcup_{\vec{x}\in S} f(\vec{x}) \cup \{MAX\}\right)$$
6. Let
$$MIN := \min\left(\bigcup_{\vec{x}\in S} f(\vec{x}) \cup \{MIN\}\right)$$
7. Save couples $\langle MAX, \alpha\rangle$ and $\langle MIN, \alpha\rangle$
8. If $\alpha > 0$ goto 3
9. Reconstruct the resulting fuzzy set from the saved couples

Under some special circumstances (hardware implementation, real-time software application, lazy programmer) it may happen that the second step brings unpleasant complications. We may avoid them by admitting only fuzzy operands with singleton cores—fuzzy numbers.

The asymptotic computational complexity of the above algorithm is $\mathcal{O}(k{\cdot}2^n)$. The good news is that the complexity is proportional to the desired resolution. From the theoretical point of view, the bad news is the exponential growth with respect to the number of distinct fuzzy operands (occurrence of crisp real numbers as operands does not make it worse, at least asymptotically). It is the penalty caused by the fact that the computation must be done globally and cannot be further decomposed.

3.4 Class G

There is still the sound kernel of the class G which withstands all our simplifying attacks. There are two remaining possibilities, both of them are very unpleasant. Either we can use some sophisticated symbolic method with all its implementation difficulties or we can waste time with an iterative search.

When performing the iterative search, we start from 1-cuts and proceed to

lower cuts, using the extremes already calculated. For each α-cut, $\alpha < 1$, we search for new extremes only on the boundary of the respective multidimensional interval and compare them to the extremes already calculated. The complexity can be of order u^n.

In the search for local extremes on the boundary of each α-cut, we can use the local extremes from the next higher α-cut as initial values. An extensive search for global extremes could be performed after several α-cuts. If new local extremes are found for this α-cut, we return to the preceding α-cuts for verification, otherwise we take the preceding results as definite and proceed to the next lower α-cut.

However, the decomposition sometimes allows to avoid the search in a high-dimensional space at all even if the expression does not belong to the class V. Let us demonstrate it on this example: the expression

$$E = ((1 - (A - 1) \cdot (A - 1)) \cdot B)_{\mathrm{CFA}}$$

with triangular fuzzy numbers $A = B$ parameterized by $\langle 0, 1, 2 \rangle$ is decomposed into three irreducible subformulas: 1, $(A - 1) \cdot (A - 1)$ and B. The first subexpression is constant (class S) , the second one belongs to the class V and the vertex algorithm can be applied to it, the third one is again in the class S. If we proceed along the α-axis in the top-down direction, we can easily determine the bounds of the corresponding α-cuts. After putting the subresults together, we obtain E with $\mathrm{Supp}\, E = (0, 2)$. This result could not be achieved by applying the vertex algorithm to the whole expression. The expression E belongs only to the class G, not to the class V, although all its irreducible components belong to the class V. What is important, the computational complexity of E is not higher than that of the class V. The conclusion is that after combining the vertex algorithm with the decomposition into irreducible expressions the region of solvable problems again slightly grows.

Let us modify the latter expression :

$$F = ((1 - (A - 1) \cdot (A - 1) + B - B + C) \cdot B)_{\mathrm{CFA}} \ .$$

Although it is equivalent to

$$((1 - (A - 1) \cdot (A - 1) + C) \cdot B)_{\mathrm{CFA}} \ ,$$

here the previous trick cannot be applied since F is irreducible. It seems we will have to explore a continuous three-dimensional space. Luckily, we know that the expression is increasing with respect to B and C. Hence, instead of a time-consuming search in the whole volume of this three-dimensional interval, we have to search along its edges in direction A only. We did not eliminate the search in the continuous space, but we reduced the problem by two dimensions.

4 Conclusion

The performance of the constrained fuzzy arithmetic with an acceptable efficiency is a highly nontrivial task. We suggested several hints that simplify the

calculations for some classes of expressions. An efficient implementation would require special procedures for various types of expressions like in symbolic integration. The whole task could lead to programs using an approach similar to that of computer algebra systems.

Appendix: Implementation of the vertex algorithm

For the implementation we have chosen Matlab 5.3. If the computation is fast enough in Matlab, it should be sufficiently fast everywhere. Matlab eases the visualization of fuzzy sets, it also allows very elegant processing of vertices of a multidimensional interval by its vector operators.

The whole system consists of several m-files which can be downloaded in the zipped form from

`http://cs.felk.cvut.cz/~zabokrtz/cfa`

or

`ftp://cmp.felk.cvut.cz/pub/cmp/articles/navara/cfa`

The most important are the following:

- `view.m` enables a user-friendly parameterization of 6 linguistic variables denoted by $a, b, \ldots, f$. Single-arrow buttons change the corresponding parameter by 0.1, double-arrow buttons increase or decrease it by 1.
- `cfa.m` serves for fuzzy expression evaluation. The user can observe the result in the classical fuzzy arithmetic as well as in the constrained fuzzy arithmetic. He/she can also change the number of α-cuts. The HTML document which includes the expression, the images with all operands, and with both resulting curves can be automatically exported. The outcoming file `index.html` with all related JPEGs is saved in the subdirectory `export`.

When solving the generally problematic second step of the algorithm—finding the global optimum in multidimensional interval given by cores of operands—we use Matlab standard function `fmins`.

A sample session in Matlab may look like this:

```
>> cd c:\Projects\Cfa

>> view
>> view
```

...several views for parameterization can be used simultaneously in order to make all used operands visible. After executing them, the user should push the button Init in order to initiate the parameterization.

>> `cfa` ... and now the user is free to make experiments.

If it is required to compute and display either the constrained or the standard arithmetic result, it is possible to set variables `constrained` or `unconstrained` to 0 (or inversely to 1, when turning it on) in the Matlab command line. The following statement switches off the constrained half of a process:
>> `constrained=0`

When it is desirable to change the number of α-cuts, we can set the variable `cutsN`.
>> `cutsN=8`

Tests of speed gave the following results: For 10 α-cuts and 6 different fuzzy operands, we have $10 \cdot 2^6$ evaluations in the cycle between steps 3 and 8, altogether with the overhead this takes not more than 4 seconds in Matlab on a 400 MHz PC. Unfortunately, during the second step of the algorithm—the exploration of the central core "hyperblock"—the `fmins` function takes more than 1 minute. This is still the bottleneck of our implementation. Fortunately, it does not degrade the algorithm, it only shows that Matlab function `fmins` is not feasible for our purpose. Using the function `minimize` in Maple Release 5, we get the output value in less than 0.5 second.

We are aware that the "unconstrained" procedure is not very optimized in our implementation. It is included only for the sake of comparison of outputs.

References

[1] Kaufmann, A., Gupta, M.M.: *Introduction to Fuzzy Arithmetic. Theory and Applications.* International Thomson Computer Press, London, 1991.

[2] Klir, G.J.: The role of constrained fuzzy arithmetic in engineering. In: B.M. Ayyub et al. (eds.), *Uncertainty Analysis in Engineering and Sciences: Fuzzy Logic, Statistics, and Neural Network Approach,* Kluwer, Dordrecht, 1997, 1–19.

[3] Klir, G.J., Pan, Y.: Constrained fuzzy arithmetic: Basic questions and some answers. *Soft Computing* **2** (1998), No. 2, 100–108.

[4] Mareš, M.: *Computation over Fuzzy Quantities.* CRC Press, Boca Raton, 1994.

[5] Žabokrtský, Z.: *Constrained Fuzzy Arithmetic: Engineer's View.* Research Report CTU–CMP–2000–03, Center for Machine Perception, Czech Technical University, Prague, Czech Republic, 2000.

[6] Zadeh, L.A.: The concept of a linguistic variable and its applications to approximate reasoning. Part I, II, III. Inform. Sci. **8** (1975), 199–251, 301–357, **9** (1975), 43–80.