12-1-2022

Data Processing under Fuzzy Uncertainty: Towards More Accurate Algorithms

Marina Tuyako Mizukoshi
Federal University of Goias, tuyako@ufg.br

Weldon Lodwick
University of Colorado Denver, weldon.lodwick@ucdenver.edu

Martine Ceberio
The University of Texas at El Paso, mceberio@utep.edu

Olga Kosheleva
The University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-22-124

Data Processing under Fuzzy Uncertainty: Towards More Accurate Algorithms

Marina Tuyako Mizukoshi, Weldon Lodwick, Martine Ceberio, Olga Kosheleva, and Vladik Kreinovich

Abstract Data that we process comes either from measurements or from experts – or from the results of previous data processing that were also based on measurements and/or expert estimates. In both cases, the data is imprecise. To gauge the accuracy of the results of data processing, we need to take the corresponding data uncertainty into account. In this paper, we describe a new algorithm for taking fuzzy uncertainty into account, an algorithm that, for small number of inputs, leads to the same or even better accuracy than the previously proposed methods.

1 Outline

In many practical situations, our information about the values of physical quantities $x_1, \ldots, x_n$ comes from experts, and experts usually describe their estimates by using imprecise ("fuzzy") words from natural language. A natural way to describe this information in computer-understandable terms is to use fuzzy techniques. When we process this data, i.e., when we estimate the quantity y which is related to x_i by a

Marina Tuyako Mizukoshi
Federal University of Goias, Brazil, e-mail: tuyako@ufg.br

Weldon Lodwick
Department of Mathematical and Statistical Sciences, University of Colorado Denver,
1201 Larimer Street, Denver, Colorado 80204, USA, e-mail: weldon.lodwick@ucdenver.edu

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

Olga Kosheleva
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA, e-mail: olgak@utep.edu

known dependence $y = f(x_1, \ldots, x_n)$, we need to take the expert's imprecision into account.

Algorithms for computing the corresponding information about the desired quantity y are usually based on interval computations; these algorithms are described in Section 2. In general, the problem of interval computations is NP-hard, so all known feasible algorithms provide only an approximate estimate for y's information.

If we want more accurate estimates, we can, in principle, use more accurate (and more time-consuming) interval computation techniques. What we show in this paper is that for applications to data processing under fuzzy uncertainty, there are other approaches to improve the accuracy, approaches that, for small numbers of inputs, are comparable in accuracy – or even more accurate – that the currently used ones.

2 Data Processing under Fuzzy Uncertainty: Definitions, State of the Art, and Remaining Problems

Need for data processing. In many practical situations, we want to know the value of a quantity y which is difficult or even impossible to measure directly. For example, we want to know tomorrow's temperature or a distance to a faraway star. Since we cannot measure the desired quantity directly, a natural idea is to estimate it indirectly:

- find easier-to-measure-or-estimate quantities $x_1, \ldots, x_n$ that are related to y by a known relation $y = f(x_1, \ldots, x_n)$,
- estimate x_i, and
- use the resulting estimates for x_i and the known relation between y and x_i to estimate y.

This is an important case of *data processing*.

Need for data processing under fuzzy uncertainty. In many cases, the only way to estimate the values x_i is to ask experts, and expert's estimates are often given not in precise terms, but rather by using imprecise ("fuzzy") words from natural language. For example, an expert may say that the value of a quantity is very small, or that this value is close to 0.1. In this case, to find a resulting estimate for y, we need to perform data processing under such fuzzy uncertainty.

Fuzzy techniques: main idea. To process such estimates, Lotfi Zadeh proposed a technique that he called *fuzzy*; see, e.g., [1, 3, 7, 10, 11, 12]. In this technique, for each imprecise natural-language term P like "very small" and for each possible value x of the corresponding quantity, we ask an expert to estimate, on a scale from 0 to 1, the degree $\mu_P(x)$ to which this value can be described by this term. Of course, there are infinitely many possible values x, and we cannot ask infinitely many questions to an expert; so:

- we ask this question about finitely many values, and then

- we use some interpolation/extrapolation algorithm to estimate the degrees for all other values x.

The resulting function $\mu_P(x)$ is called a *membership function* or a *fuzzy set*.

Fuzzy "and"- and "or"-operations. In many cases, the expert's rules involve logical connectives like "and" and "or". For example the rule may explain what to do if the car in front is close *and* it slows down a little bit. Ideally, we could ask the expert to estimate his/her degree of confidence in all such complex statements, but there are too many such possible complex statements, and it is not feasible to ask the expert about all of them. So, we need to be able, given the degrees of confidence a and b in statements A and B, to compute the estimates for the expert's degree of confidence in statements $A \,\&\, B$ and $A \vee B$. The algorithms for computing such estimates are known as, correspondingly, "and"-operations $f_{\&}(a,b)$ (also known as t-norms) and "or"-operations $f_{\vee}(a,b)$ (also known as t-conorms). The simplest "and"- and "or"-operations are $f_{\&}(a,b) = \min(a,b)$ and $f_{\vee}(a,b) = \max(a,b)$.

Data processing under fuzzy uncertainty. Suppose that have fuzzy information $\mu_i(x_i)$ about each input x_i. What can we say about the value $y = f(x_1,\ldots,x_n)$?

A number y is a possible value of the resulting quantity if for some tuple $(x_1,\ldots,x_n)$ for which $y = f(x_1,\ldots,x_n)$, x_1 is a possible value of the first input, *and* x_2 is a possible value of the second input, ... We know the degrees $\mu_i(x_i)$ to which each value x_i is possible. Here, "for some" means "or", so if we use min to describe "and" and max to describe "or", we get the following formula

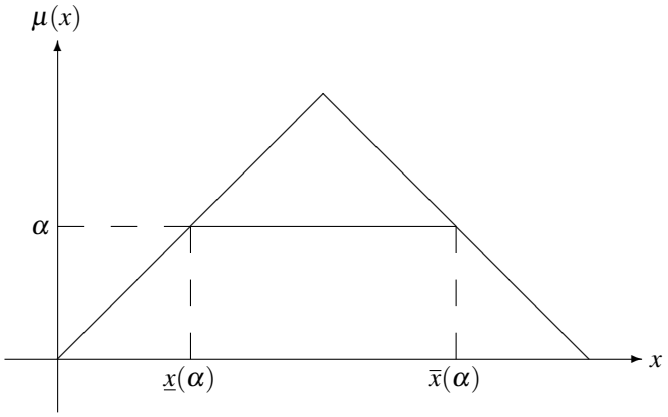
$$\mu(y) = \sup_{(x_1,\ldots,x_n):\ f(x_1,\ldots,x_n)=y} \min(\mu_1(x_1),\ldots,\mu_n(x_n)). \qquad (1)$$

This formula was first described by Zadeh and is thus known as *Zadeh's extension principle*.

Data processing under fuzzy uncertainty: reduction to interval computations. It is known that the formula (1) becomes simpler if instead of the original membership functions $\mu_i(x_i)$, we consider its α-*cuts*

$$[\underline{x}_i(\alpha), \overline{x}_i(\alpha)] \stackrel{\text{def}}{=} \{x_i : \mu_i(x_i) \geq \alpha\}$$

corresponding to different $\alpha \in (0,1]$.



It is convenient to denote each α-cut by

$$\mathbf{x}_i(\alpha) \stackrel{\text{def}}{=} [\underline{x}_i(\alpha), \overline{x}_i(\alpha).$$

In these notations, each α-cut $\mathbf{y}(\alpha) = \{y : \mu(y) \geq \alpha\}$ corresponding to y is equal to

$$\mathbf{y}(\alpha) = f(\mathbf{x}_1(\alpha), \ldots, \mathbf{x}_n(\alpha)),$$

where for each tuple of sets $X_1, \ldots, X_n$, the y-range $f(X_1, \ldots, X_n)$ is defined as

$$f(X_1, \ldots, X_n) \stackrel{\text{def}}{=} \{f(x_1, \ldots, x_n) : x_1 \in X_1, \ldots, x_n \in X_n\}.$$

Usually, membership functions corresponding to terms like "very small" or "close to 0.1" are first non-strictly increasing and then non-strictly decreasing. In this case, each α-cut is an interval, and the problem of computing each α-cut $\mathbf{y}(\alpha)$ becomes the problem of computing the set $f(X_1, \ldots, X_n)$ for n intervals $X_i = \mathbf{x}_i(\alpha)$. The problem of computing the y-range $f(X_1, \ldots, X_n)$ for intervals $X_1, \ldots, X_n$ is known as the problem of *interval computations*; see, e.g., [2, 5, 6, 8].

Comment. To reconstruct the membership function exactly, we need to know the α-cuts corresponding to all possible values α. Of course, there are infinitely real numbers α in the interval $(0, 1]$, but at any period of time, we can only perform finitely many computations. So, in practice, we compute the α-cuts for finitely many values $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_m \leq 1$. We try to select these values α_i to provide a good approximation to the original membership function, so, to avoid large gaps, we make sure that the differences $\alpha_{k+1} - \alpha_k$ between the consequent values of α are small.

Interval computations: a brief reminder. In general, the interval computation problem is NP-hard (see, e.g., [4]). This means, crudely speaking, that no feasible algorithm can always compute the exact y-range $f(X_1, \ldots, X_n)$ for intervals

$X_i = [\underline{x}_i, \overline{x}_i]$. Thus, each feasible algorithm provides only an approximation to the y-range.

The usual algorithms for computing the y-range can be explained in several steps. The first step is the fact that when the function $f(x_1, x_2)$ corresponds to a simple arithmetic operation like addition, subtraction, multiplication, and division, the range of the result can be explicitly computed:

$$[\underline{x}_1, \overline{x}_1] + [\underline{x}_2, \overline{x}_2] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2]; \;\; [\underline{x}_1, \overline{x}_1] - [\underline{x}_2, \overline{x}_2] = [\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2];$$

$$[\underline{x}_1, \overline{x}_1] \cdot [\underline{x}_2, \overline{x}_2] = [\min(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2), \max(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2)];$$

$$\frac{1}{[\underline{x}_1, \overline{x}_1]} = \left[\frac{1}{\overline{x}_1}, \frac{1}{\underline{x}_1}\right] \text{ if } 0 \notin [\underline{x}_1, \overline{x}_1]; \text{ and } \frac{[\underline{x}_1, \overline{x}_1]}{[\underline{x}_2, \overline{x}_2]} = [\underline{x}_1, \overline{x}_1] \cdot \frac{1}{[\underline{x}_2, \overline{x}_2]}.$$

These formulas are known as *interval arithmetic* (sometimes called *standard interval arithmetic*, to distinguish it from alternative techniques for processing intervals).

The second step is related to the fact that in a computer, arithmetic operations are the only ones that are hardware supported. Thus, in a computer, every computation is actually a sequence of arithmetic operations. It is known that if we replace each arithmetic operation with numbers with the corresponding operation of interval arithmetic, then we get an *enclosure* $\mathbf{Y}$ for the desired range $\mathbf{y}$, i.e., a set $\mathbf{Y}$ for which $\mathbf{y} \subseteq \mathbf{Y}$. This procedure of replacing each arithmetic operation with the corresponding operation of interval arithmetic is known as *straightforward interval computations.* (This procedure is also known as *naive interval computations*, since it is sometimes used – usually unsuccessfully – by people who naively think that this is what interval computations are about – not realizing that they are missing an important part of interval techniques.)

The third step is the so-called *centered form*, according to which the desired range $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is enclosed by the following interval:

$$\mathbf{y} \subseteq \widetilde{y} + \sum_{i=1}^{n} \mathbf{d}_i \cdot [-\Delta_i, \Delta_i],$$

where

$$\Delta_i \stackrel{\text{def}}{=} \frac{\overline{x}_i - \underline{x}_i}{2}$$

is the half-width (= radius) of the i-th interval, $\widetilde{y} \stackrel{\text{def}}{=} f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, where

$$\widetilde{x}_i \stackrel{\text{def}}{=} \frac{\underline{x}_i + \overline{x}_i}{2}$$

is the midpoint of the i-th interval, and $\mathbf{d}_i$ is the result of applying straightforward interval computations to estimate the range of the i-th partial derivative of f on the intervals $\mathbf{x}_i$:

$$\mathbf{d}_i \supseteq \frac{\partial f}{\partial x_i}(\mathbf{x}_1, \ldots, \mathbf{x}_n).$$

(There is also an additional idea – that we can simplify computations if the function $f(x_1,\ldots,x_n)$ is monotonic with respect to some of the variables – as was, e.g., the case of arithmetic operations.)

Since, as we have mentioned, the problem of computing the exact y-range is NP-hard, the centered form leads, in general, to an approximation to the actual y-range. What if we want a more accurate estimate? The approximation error of the centered form approximation can be estimated if we take into account that the centered form is, in effect, based on the linear terms in the Taylor expansion of the function $f(x_1,\ldots,x_n)$ – and indeed, for linear functions $f(x_1,\ldots,x_n)$, the centered form formula leads to the exact range. Thus, the approximation error of this approximation is of the same order of magnitude as the largest terms that we ignore in this approach – i.e., quadratic terms, terms proportional to the products $\Delta_i \cdot \Delta_j$. From this viewpoint, to get a more accurate estimate, we can:

- divide (bisect) one of the intervals into two equal parts,
- estimate the range corresponding to each of these parts, and then
- take the union of the corresponding y-ranges.

After bisection, the width of one of the intervals decreases in half, so some terms in the general quadratic expression become smaller, and the approximation error decreases.

Remaining problems. The current algorithm requires a certain computation time and leads to a certain accuracy.

- In some cases, we need the results faster – and it is OK if the accuracy is slightly lower; in this case, we can use an alternative algorithm described in [9].
- In other cases, we are interested in higher accuracy – even if it takes more computation time. This is a problem with which we deal in this paper.

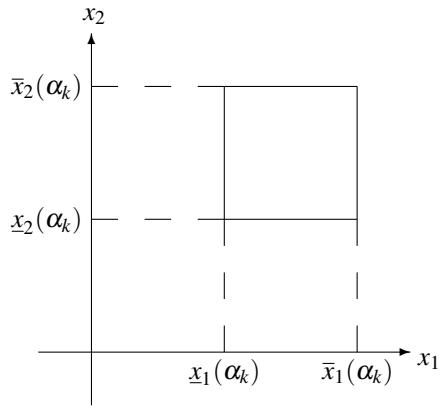
What we do in this paper. As we have mentioned, a general way to get more accurate estimates is to apply bisection. In this paper, we propose another way of increasing accuracy, which is specifically tailored to the case of fuzzy data processing.

3 New Algorithm: Motivations, Description, and Comparative Analysis

Motivations. For each α_k, the corresponding y-α-cut $\mathbf{y}(\alpha_k)$ is equal to the y-range $f(B(\alpha_k))$ of the function $f(x_1,\ldots,x_n)$ over the corresponding x-range box

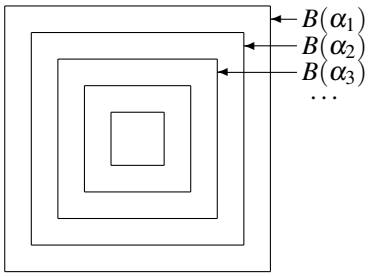
$$B(\alpha_k) \stackrel{\text{def}}{=} [\underline{x}_1(\alpha_k),\overline{x}_1(\alpha_k)] \times \ldots \times [\underline{x}_n(\alpha_k),\overline{x}_n(\alpha_k)].$$

For example, for $n = 2$, the x-range box has the following form:



From the definition of the α-cut, we can conclude that the x-ranges corresponding to different values α_k are getting narrower as α increases:

$$B(\alpha_1) \supseteq B(\alpha_2) \supseteq \ldots \supseteq B(\alpha_m).$$

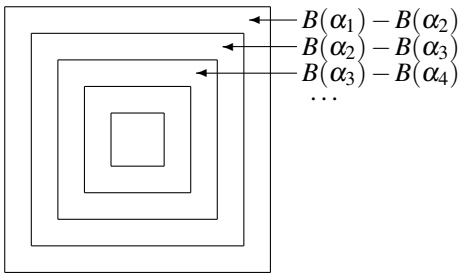


Because of this, each x-range can be represented as a union of the differences:

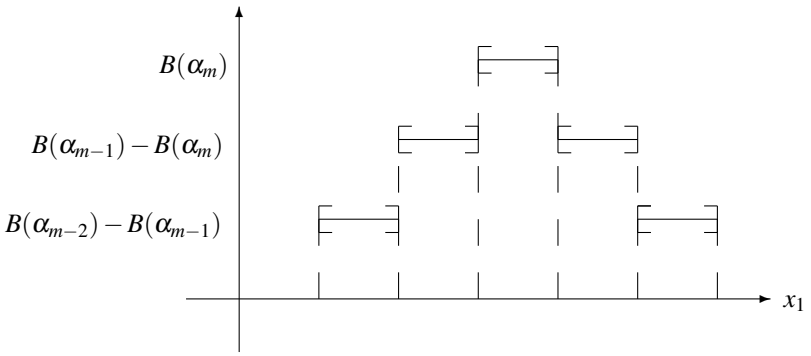
$$B(\alpha_k) = B(\alpha_m) \cup (B(\alpha_{m-1}) - B(\alpha_m)) \cup \ldots \cup (B(\alpha_k) - B(\alpha_{k+1})),$$

where, as usual, $B(\alpha_j) - B(\alpha_{j+1})$ denotes the difference between the two sets

$$B(\alpha_j) - B(\alpha_{j+1}) \stackrel{\text{def}}{=} \{a : a \in B(\alpha_j) \text{ and } a \notin B(\alpha_{j+1})\}.$$



For example, for $n = 1$, we get:



Thus, the desired y-range $\mathbf{y}(\alpha_k)$ can be represented as the union of the y-ranges corresponding to the component sets:

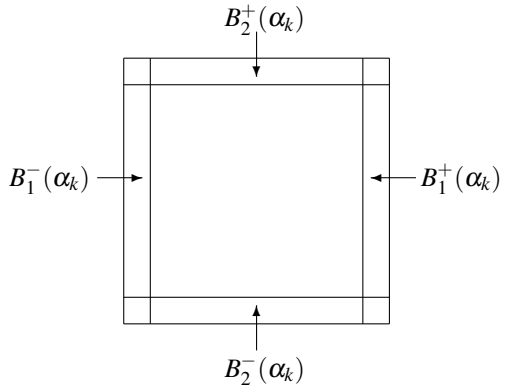
$$f(B(\alpha_k)) = f(B(\alpha_m)) \cup f(B(\alpha_{m-1}) - B(\alpha_m)) \cup \ldots \cup f(B(\alpha_k) - B(\alpha_{k+1})). \quad (2)$$

Each difference $B(\alpha_k) - B(\alpha_{k+1})$ is the union of (somewhat overlapping) $2n$ boxes corresponding to $i = 1, \ldots, n$:

$$B_i^-(\alpha_k) = [\underline{x}_1(\alpha_k), \overline{x}_1(\alpha_k)] \times \ldots [\underline{x}_{i-1}(\alpha_k), \overline{x}_{i-1}(\alpha_k)] \times [\underline{x}_i(\alpha_k), \underline{x}_i(\alpha_{k+1})] \times$$
$$[\underline{x}_{i+1}(\alpha_k), \overline{x}_{i+1}(\alpha_k)] \ldots \times \ldots \times [\underline{x}_n(\alpha_k), \overline{x}_n(\alpha_k)] \quad (3)$$

and

$$B_i^+(\alpha_k) = [\underline{x}_1(\alpha_k), \overline{x}_1(\alpha_k)] \times \ldots [\underline{x}_{i-1}(\alpha_k), \overline{x}_{i-1}(\alpha_k)] \times [\overline{x}_i(\alpha_{k+1}), \overline{x}_i(\alpha_k)] \times$$
$$[\underline{x}_{i+1}(\alpha_k), \overline{x}_{i+1}(\alpha_k)] \ldots \times \ldots \times [\underline{x}_n(\alpha_k), \overline{x}_n(\alpha_k)]. \quad (4)$$



Thus, the y-range corresponding to each difference $B(\alpha_k) - B(\alpha_{k+1})$ is equal to the union of y-ranges corresponding to the component boxes:

$$f(B(\alpha_k) - B(\alpha_{k+1})) =$$

$$f(B_1^-(\alpha_k)) \cup f(B_1^+(\alpha_k)) \cup \ldots \cup f(B_n^-(\alpha_k)) \cup f(B_n^+(\alpha_k)). \quad (5)$$

So, we can compute the y-ranges over these component boxes, and then take the union.

In each of the component boxes (3) and (4), the width of one of the sides is much narrower than in the box $B(\alpha_k)$. In this case, as we have mentioned, we get a more accurate result. This leads us to the following algorithm.

Algorithm.

- We compute the y-range $f(B_m)$ over the box

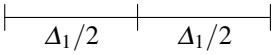
$$B(\alpha_m) \stackrel{\text{def}}{=} [\underline{x}_1(\alpha_m), \overline{x}_1(\alpha_m)] \times \ldots \times [\underline{x}_n(\alpha_m), \overline{x}_n(\alpha_m)]. \quad (6)$$

- For each k from 1 to $m-1$ and for each i from 1 to n, we use the centered form (or any other interval computation technique) to estimate the ranges $f(B_1^-(\alpha_k))$ and $f(B_1^+(\alpha_k))$ over the boxes (3) and (4).
- For each k from 1 to $m-1$, we compute the range $f(B(\alpha_k) - B(\alpha_{k+1}))$ by using the formula (5).
- Finally, for each k from 1 to $m-1$, we compute the y-range $\mathbf{y}(\alpha_k) = f(B(\alpha_k))$ by using the formula (2).

Is this algorithm better than bisection? In the above-described usual algorithm for data processing under interval uncertainty, we apply interval computations m times – as many times as there are different values α_k. In the new algorithm, we apply it $1 + (m-1) \cdot (2n)$ times. So, clearly, the new algorithm takes longer time: approximately $2n$ times longer.

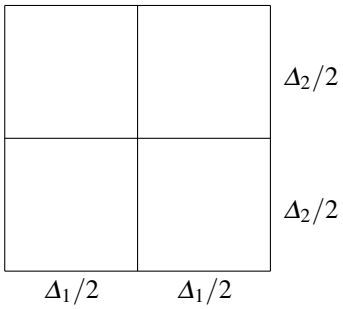
Similarly, if we use bisection, we get more accurate results, but also at the expense of a longer computation time. So, the question is which of these two algorithms leads to more accurate estimates if we use the same computation time. Let us analyze this question for different values n.

Case when $n = 1$: the new algorithm is more accurate. For $n = 1$, the new algorithm requires $2n = 2$ times longer than the original centered form. So, we need to compare with the case when we apply bisection once – in this case we need two applications of centered form, i.e., we also increase the computation time by a factor of 2.



The approximation error of the original centered form is proportional to Δ_1^2. If we bisect, the width decreases to $\Delta_1/2$, so the approximation error of the bisection result – which is proportional to $(\Delta_1/2)^2 = \Delta_1^2/4$ – decreases by a factor of 4. On the other hand, for the new algorithm, the approximation error is proportional to the square of much smaller widths like $\underline{x}_i(\alpha_k) - \underline{x}_i(\alpha_{k+1})$ which are, in general, m times smaller than the original width. So, with the new algorithm, the approximation error decreases by a factor of m^2, which for usual $m = 7 \pm 2$, is much better than for bisection. Thus, for $n = 1$, the new method is much more accurate than bisection.

Case when $n = 2$: both algorithms are of the same accuracy. In this case, the new algorithm requires $2n = 4$ times longer than the original centered form. So, we need to compare with the case when we apply bisection twice – then we also increase the computation time by a factor of 4.



In this case, we bisect both input intervals, so both widths are decreased by a factor of 2, and thus, all products $\Delta_i \cdot \Delta_j$ involved in our estimate for the approximation error also decrease by a factor of 4. Thus, for bisection, the new approximation error is 4 times smaller than for the original centered form.

In the new method, on each of the four estimates, only one side has the original width, the other side is much smaller. Thus, instead of the four terms $\Delta_i \cdot \Delta_j$ ($i, j = 1, 2$) forming the original approximation error, only one terms remains the same size – all others are much smaller. So, the new method also decreases the approximation error by a factor of 4. Thus, for $n = 2$, the two methods have comparable accuracy.

Case when $n = 3$: both algorithms are of the same accuracy. In this case, the new algorithm requires $2n = 6$ times longer than the original centered form. So, we need to compare with the case when we apply bisection and get 6 smaller boxes. This way, for all resulting boxes, we can get the first two coordinates divided by two. However, we do not have enough boxes to makes sure that for all the boxes, all the widths are divided by 2 – this would require 8 boxes. Thus, at least for some of the smaller boxes, we will have the third coordinate the same size as before.

So:

- for 4 terms $\Delta_i \cdot \Delta_j$ corresponding to $i, j \le 2$, we get 1/4 of the original size,
- for $2 \cdot 2 = 4$ terms $\Delta_i \cdot \Delta_3$ and $\Delta_3 \cdot \Delta_i$, $i \le 2$, we get 1/2 of the original size, and
- the term Δ_3^2 remains the same.

Thus, the overall approximation error – which was originally consisting of $3 \times 3 = 9$ original-size terms $\Delta_i \cdot \Delta_j$ – is now reduced to

$$4 \cdot (1/4) + 4 \cdot (1/2) + 1 = 4$$

times the original product term. So, for bisection, the approximation error is smaller by a factor of $9/4 = 2.25$.

In the new method, for each box, we only have two sides of the original width, the third side is much narrower. So, out of 9 products, only 4 products corresponding to the original-width sides remains the same, the rest become much smaller and can, thus, safely be ignored. Thus, for $n = 3$, the new method also decreases the approximation error by the same factor of $9/4 = 2.25$, so the two methods have comparable accuracy.

Case of larger n: bisection is more accurate. In general, if we bisect the box over v variables, we get 2^v different boxes – and thus, 2^v times longer computations. To compare with the new algorithm that requires $2n$ times more computations, we need to select v for which $2^v = 2n$, i.e., $v = \log_2(2n)$. Here, $v \ll n$. In this case:

- for v^2 pairs of bisected variables, the product $\Delta_i \cdot \Delta_j$ is decreased by a factor of 4;
- for $2v \cdot (n - v)$ pairs of a bisected and non-bisected variable, the product $\Delta_i \cdot \Delta_j$ is decreased by a factor of 2; and
- for $(n - v)^2$ pairs of non-bisected variables, the product $\Delta_i \cdot \Delta_j$ remains the same.

Thus, after this bisection, instead of the sum of n^2 products of the original size, have the sum proportional to

$$\frac{1}{4} \cdot v^2 + \frac{1}{2} \cdot 2 \cdot v \cdot (n - v) + (n - v)^2 = \frac{1}{4} \cdot v^2 + v \cdot n - v^2 + n^2 - 2 \cdot v \cdot n + v^2 =$$

$$n^2 - v \cdot n + \frac{1}{4} \cdot v^2 = n^2 - \log_2(2n) \cdot n + O(\log^2(n)).$$

On the other hand, for the new algorithm, for each box, $n - 1$ sizes are the same, and one is much smaller, so we reduce the sum of n terms to the sum of $(n - 1)^2$ terms, i.e., to

$$(n - 1)^2 = n^2 - 2 \cdot n + 1.$$

For large n, we have $\log_2(2n) > 2$, and thus, bisection leads to more accurate results.

This advantage starts with the case when $n = 4$. In this case, the new algorithm requires $2n = 8$ uses of the centered form. During this times, we can bisect each of the first 3 inputs – this will also lead to $2^3 = 8$ boxes and, thus, 8 uses of the centered form. Without loss of generality, let us assume that we bisect the first three inputs. Then:

- for 9 terms $\Delta_i \cdot \Delta_j$ corresponding to $i, j \le 3$, we get 1/4 of the original size,
- for $2 \cdot 3 = 6$ terms $\Delta_i \cdot \Delta_4$ and $\Delta_4 \cdot \Delta_i$, $i \le 2$, we get 1/2 of the original size, and
- the term Δ_3^2 remains the same.

Thus, the overall approximation error – which was originally consisting of $4 \times 4 = 16$ original-size terms $\Delta_i \cdot \Delta_j$ – is now reduced to

$$9 \cdot (1/4) + 6 \cdot (1/2) + 1 = 6.25$$

times the original product term. So, for bisection, the approximation error is smaller by a factor of $16/6.25 = 2.56$.

On the other hand, in the new method, for each box, we only have 3 sides of the original width, the 4th side is much narrower. So, out of 16 products, only $3^3 = 9$ products corresponding to the original-width sides remains the same, the rest become much small and can, thus, safely be ignored. Thus, for $n = 4$, the new method also decreases the approximation error by the factor of $16/9 \approx 1.78$, not as good as bisection.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes), and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References

1. R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.
2. L. Jaulin, M. Kiefer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control, and Robotics*, Springer, London, 2001.
3. G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

4. V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.
5. B. J. Kubica, *Interval Methods for Solving Nonlinear Contraint Satisfaction, Optimization, and Similar Problems: from Inequalities Systems to Game Solutions*, Springer, Cham, Switzerland, 2019.
6. G. Mayer, *Interval Analysis and Automatic Result Verification*, de Gruyter, Berlin, 2017.
7. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.
8. R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM, Philadelphia, 2009.
9. H. T. Nguyen, O. Kosheleva, and V. Kreinovich, "Data processing under fuzzy uncertainty: towards more efficient algorithms", *Proceedings of the 2022 IEEE World Congress on Computational Intelligence IEEE WCCI'2022*, Padua, Italy, July 18–23, 2022.
10. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
11. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
12. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.