Fuzzy Data Processing Beyond Min t-Norm

Andrzej Pownuk[1], Vladik Kreinovich[1], and
Sonsgak Sriboonchitta[2]
[1]Computational Science Program
University of Texas at El Paso
El Paso, TX 69968, USA
ampownuk@utep.edu, vladik@utep.edu
[2]Faculty of Economics
Chiang Mai University
Chiang Mai 50200 Thailand
songsakecon@gmail.com

Abstract

Usual algorithms for fuzzy data processing – based on the usual form of Zadeh's extension principle – implicitly assume that we use the min "and"-operation (t-norm). It is known, however, that in many practical situations, other t-norms more adequately describe human reasoning. It is therefore desirable to extend the usual algorithms to situations when we use t-norms different from min. Such an extension is provided in this chapter.

1 Need for Fuzzy Data Processing

Need for data processing. In many real-life situations, we are interested in the value of a quantity y which are difficult (or even impossible) to measure directly. For example, we may be interested in the distance to a faraway star, in the amount of oil in a given well, or in tomorrow's weather.

Since we cannot measure the quantity y directly, the way to estimate this value is to measure related easier-to-measure quantities $x_1, \ldots, x_n$, and then to use the known relation $y = f(x_1, \ldots, x_n)$ between the desired quantity y and the quantities x_i to estimate y; see, e.g, [8].

For example, since we cannot directly measure the distance to a faraway star, we can measure the directions x_i to this star in two different seasons, when the Earth is on the opposite sides of its Solar orbit, and then use trigonometry to find the desired distance. Since we cannot directly measure the amount y of oil in the well, we can measure the results of artificially set seismic waves propagating through the corresponding region, and then use the known equations of seismic wave propagation to compute y. Since we cannot directly measure tomorrow's temperature y, to estimate y, we measure temperature, moisture, wind speed, and other meteorological characteristics in the vicinity of our area, and use the known equations of aerodynamics and thermal physics to predict tomorrow's weather.

The computation of $y = f(x_1, \ldots, x_n)$ based on the results of measuring x_i is known as *data processing.*

In some cases, the data processing algorithm $f(x_1, \ldots, x_n)$ consists of direct application of a formula, but in most cases, we have a rather complex algorithm – such as solving a system of partial differential equations.

Need for fuzzy data processing. In some cases, instead of the measurement results $x_1, \ldots, x_n$, we have expert estimates for the corresponding quantities. Experts can rarely describe their estimates by exact numbers, they usually describe their estimates by using imprecise ("fuzzy") words from natural language. For example, an expert can say that the temperature is *warm*, or that the temperature is *around 20.*

To process such imprecise natural-language estimates, L. Zadeh designed the technique of *fuzzy logic*; see, e.g., [3, 7, 10]. In this technique, each imprecise word like "warm" is described by assigning, to each possible value of the temperature x, the degree to which the expert believes that this particular temperature is warm.

This degree can be obtained, e.g., by asking an expert to mark, on a scale from 0 to 10, where 0 means no warm and 10 means warn, how much x degrees means warm. Then, if for some temperature x, the expert marks, say, 7 on a scale from 0 to 10, we assign the degree $\mu(x) = 7/10$.

The function that assigns, to each possible value of a quantity x, the degree $\mu(x)$ to which the quantity satisfies the expert's estimate, is known as the *membership function.*

If for each of the inputs $x_1, \ldots, x_n$, we only have expert estimates, then what can we say about $y = f(x_1, \ldots, x_n)$? Computing appropriate estimates for y is known as *fuzzy data processing.*

To perform fuzzy data processing, we need to deal with propositional connectives such as "and" and "or". In fuzzy data processing, we know that $y = f(x_1, \ldots, x_n)$, and we have fuzzy information about each of the inputs x_i. Based on this information, we need to estimate the corresponding degrees of confidence in different values y.

A value y is possible 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 variable, *and* x_2 is a possible value of the second variable, ..., *and* x_n is a possible value of the n-th variable. So, to describe the degree to which each value y is possible, we need to be able to describe our degrees of confidence in statements containing propositional connectives such as "and" and "or".

Dealing with "and"- and "or" in fuzzy logic is not as easy as in the 2-valued case. In the traditional 2-valued logic, when each statement is either true or false, dealing with "and" and "or" connectives is straightforward: if we know the truth values of two statements A and B, then these truth values uniquely determine the truth values of the propositional combinations $A \,\&\, B$ and $A \vee B$.

In contrast, when we deal with degrees of confidence, the situation is not as straightforward. For example, for a fair coin, we have no reason to be more confident that it will fall head or tail. Thus, it makes sense to assume that the expert's degrees of confidence $a = d(A)$ and $b = d(B)$ in statement $A=$"coin falls head" and $B=$"coin falls tail" are the same: $a = b$.

Since a coin cannot at the same time fall head and tail, the expert's degree of belief that both A and B happen is clearly 0: $d(A \,\&\, B) = 0$. On the other hand, if we take $A' = B' = A$, then clearly $A' \,\&\, B'$ is equivalent to A and thus, $d(A' \,\&\, B') = d(A) > 0$. This is a simple example where for two different pairs of statements, we have the same values of $d(A)$ and $d(B)$ but different values of $d(A \,\&\, B)$:

- in the first case, $d(A) = d(B) = a$ and $A(A \,\&\, B) = 0$, while
- in the second case, $d(A') = d(B') = a$, but $d(A' \,\&\, B') = a > 0$.

This simple example shows that, in general, the expert's degree of confidence in a propositional combination like $A \,\&\, B$ or $A \vee B$ is not uniquely determined by his/her degrees of confidence in the statements A and B. Thus, ideally, in addition to eliciting, from the experts, the degrees of confidence in all basic statements $A_1, \ldots, A_n$, we should also elicit from them degrees of confidence in all possible propositional combinations. For example, for each subset

$$I \subseteq \{1, \ldots, n\},$$

we should elicit, from the expert, his/her degree of confidence in a statement $\&_{i \in I} A_i$.

The problem is that there are 2^n such statement (and even more if we consider different propositional combinations). Even of a small size knowledge base, with $n = 30$ statements, we thus need to ask the expert about $2^{30} \approx 10^9$ different propositional combinations – this is clearly not practically possible.

Need for "and"- and "or"-operations. Since we cannot elicit the degrees of confidence for all propositional combinations from an expert, it is therefore desirable to estimate the degree of confidence in a propositional combination like $A \,\&\, B$ or $A \vee B$ from the expert's degrees of confidence $a = d(A)$ and $b = d(B)$ in the original statements A and B – knowing very well that these are *estimates*, not exactly the original expert's degrees of confidence in the composite statements.

The corresponding estimate for $d(A \,\&\, B)$ will be denoted by $f_\&(a, b)$. It is known as an *"and"-operation*, or, alternatively, as a *t-norm.* Similarly, the estimate for $d(A \vee B)$ will be denoted by $f_\vee(a, b)$. This estimate is called an *"or"-operation*, or a *t-conorm.*

There are reasonable conditions that these operations should satisfy. For example, since "A and B" means the same as "B and A", it makes sense to require that the estimates for $A \,\&\, B$ and $B \,\&\, A$ are the same, i.e., that $f_\&(a,b) = f_\&(b,a)$ for all a and b. Similarly, we should have $f_\vee(a,b) = f_\vee(b,a)$ for all a and b. The fact that $A \,\&\, (B \,\&\, C)$ means the same as $(A \,\&\, B) \,\&\, C$ implies that the corresponding estimates should be the same, i.e., that we should have $f_\&(a, f_\&(b,c)) = f_\&(f_\&(a,b), c)$ for all a, b, and c. There are also reasonable monotonicity requirements.

There are many different "and"- and "or"-operations that satisfy all these requirements. For example, for "and", we have the min-operation $f_\&(a,b) = \min(a,b)$. We also have an *algebraic product* operation $f_\&(a,b) = a \cdot b$. In addition, we can consider general *Archimedean operations*

$$f_\&(a,b) = f^{-1}(f(a) \cdot f(b)),$$

for some strictly increasing continuous function $f(x)$.

Similarly, for "or", we have the max-operation $f_vee(a,b) = \max(a,b)$, we have *algebraic sum*

$$f_\vee(a,b) = a + b - a \cdot b = 1 - (1-a) \cdot (1-b),$$

and we have general Archimedean operations

$$f_\vee(a,b) = f^{-1}(f(a) + f(b) - f(a) \cdot f(b)).$$

Which "and" and "or"-operations should we select? Our objective is to describe the expert's knowledge. So, it is reasonable to select "and"- and "or"-operations which most adequately describe the reasoning of this particular expert (or this particular group of experts).

Such a selection was first made by Stanford researchers who designed the world's first expert system MYCIN – for curing rare blood diseases; see, e.g., [1]. It is interesting to mention that when the researchers found the "and"- and "or"-operations that best describe the reasoning of medical experts, they thought that they have found general laws of human reasoning. However, when they tried the same operations on another area of knowledge – geophysics – it turned out that different "and'- and "or'-operations are needed.

In hindsight, this difference make perfect sense: in medicine, one has to be very cautious, to avoid harming a patient. You do not start a surgery unless you are absolutely sure that the diagnosis is right. If a doctor is not absolutely sure, he or she recommends additional tests. In contrast, in geophysics, if you have a reasonable degree of confidence that there is oil in a given location, you dig a well – and you do not wait for a 100% certainty: a failed well is an acceptable risk.

What this experience taught us is that in different situations, different "and"- and "or"-operations are more adequate.

Let us apply "and"- and "or"-operation to fuzzy data processing. Let us go back to fuzzy data processing. To find the degree to which y is a possible value, we need to consider all possible tuples $(x_1, \ldots, x_n)$ for which $y = f(x_1, \ldots, x_n)$. The value y is possible if one of these tuples is possible, i.e., either the first tuple is possible *or* the second tuple is possible, etc.

Once we know the degree $d(x)$ to which each tuple $x = (x_1, \ldots, x_n)$ is possible, the degree to which y is possible can then be obtained by applying "or"-operation $f_\vee(a, b)$ to all these degrees:

$$d(y) = f_\vee\{d(x_1, \ldots, x_n) : f(x_1, \ldots, x_n)\}.$$

Usually, there are infinitely many such tuples x for which $f(x) = y$. For most "or"-operations, e.g., for

$$f_\&(a, b) = a + b - a \cdot b = 1 - (1 - a) \cdot (1 - b),$$

if we combine infinitely many terms, we get the same meaningless value 1. In effect, the only widely used operation for which this is not happening is $f_\vee(a, b) = \max(a, b)$. Thus, it is makes sense to require that

$$d(y) = \max\{d(x_1, \ldots, x_n) : f(x_1, \ldots, x_n) = y\}.$$

To fully describe this degree, we thus need to describe the degree $d(x_1, \ldots, x_n)$ to which a tuple $x = (x_1, \ldots, x_n)$ is possible. A tuple $(x_1, \ldots, x_n)$ is possible if x_1 is a possible values of the first variable, *and* x_2 is a possible of the second variable, etc.

We know the expert's degree of confidence that x_1 is a possible value of the first variable: this described by the corresponding membership function $\mu_1(x_1)$. Similarly, the expert's degree of confidence that x_2 is a possible value of the second variable is equal to $\mu_2(x_2)$, etc. Thus, to get the degree to which the tuple $(x_1, \ldots, x_n)$ is possible, we need to use an appropriate "and"-operation $f_\&(a, b)$ to combine these degrees $\mu_i(x_i)$:

$$d(x_1, \ldots, x_n) = f_\&(\mu_1(x_1), \ldots, \mu_n(x_n)).$$

Substituting this expression into the above formula for $\mu(y) = d(y)$, we arrive at the following formula.

General form of Zadeh's extension principle. If we know the the quantity y is related to the quantities $x_1, \ldots, x_n$ by a relation $y = f(x_1, \ldots, x_n)$, and we know the membership functions $\mu_i(x_i)$ that describe each inputs x_i, then the resulting membership function for y takes the form

$$\mu(y) = \max\{f_\&(\mu_1(x_1), \ldots, \mu_n(x_n)) : f(x_1, \ldots, x_n) = y\}.$$

This formula was originally proposed by Zadeh himself and is thus known as *Zadeh's extension principle.*

Algorithms for fuzzy data processing: what is already known and what we do in this paper. Usually, Zadeh's extension principle is applied to the situations in which we use the min "and"-operation $f_\&(a,b)=\min(a,b)$. In this case, Zadeh's extension principle takes a simplified form

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

For this case, there are efficient algorithms for computing $\mu(y)$. (We will mention these algorithms in the nest section.)

The problem is that, as it is well known, in many practical situations, other "and"-operations more adequately describe expert reasoning [3, 7]. It is therefore desirable to generalize the existing efficient algorithms for fuzzy data processing from the case of min to the general case of an arbitrary "and"-operation. This is what we do in this paper.

2 Possibility of Linearization

In most practical cases, measurement and estimation errors are relatively small. Before we get deeper into algorithms, let us notice that in the above text, we consider the generic data processing functions $f(x_1,\ldots,x_n)$.

In the general case, this is what we have to do. However, in most practical situations, the expert uncertainty is relatively small – just like measurement errors are usually relatively small, small in the sense that the squares of the measurement or estimation errors are much smaller than the errors themselves; see, e.g., [8].

Indeed, if we have a measurement or an expert estimate with accuracy 10% (i.e., 0.1), then the square of this inaccuracy is 0.01, which is much smaller than the original measurement or estimation error. If we measure or estimate with an even higher accuracy, e.g., 5% or 1%, then the quadratic term is even much more small than the measurement or estimation error.

Possibility of linearization. In such situations, instead of considering general functions $f(x_1,\ldots,x_n)$, we can expand the corresponding data processing function in Taylor series around the estimates $\widetilde{x}_1,\ldots,\widetilde{x}_n$, and keep only linear terms in this expansion – thus, ignoring the quadratic terms. As a result, the original expression

$$f(x_1,\ldots,x_n)=f(\widetilde{x}_1+\Delta x_1,\ldots,\widetilde{x}_n+\Delta x_n),$$

where we denoted $\Delta x_i \stackrel{\text{def}}{=} x_i-\widetilde{x}_i$, is replace by its linear approximation:

$$f(x_1,\ldots,x_n)=f(\widetilde{x}_1+\Delta x_1,\ldots,\widetilde{x}_n+\Delta x_n)\approx c_0+\sum_{i=1}^n c_i\cdot\Delta x_i,$$

where $c_0 \stackrel{\text{def}}{=} f(\widetilde{x}_1,\ldots,\widetilde{x}_n)$ and $c_i \stackrel{\text{def}}{=} \dfrac{\partial f}{\partial x_i}_{|x_i=\widetilde{x}_i}$. Substituting $\Delta x_i=x_i-\widetilde{x}_i$ into the above formula, we get a linear dependence

$$f(x_1,\ldots,x_n)=a_0+\sum_{i=1}^n c_i\cdot x_i,$$

where $a_0 \stackrel{\text{def}}{=} c_0 - \sum\limits_{i=1}^{n} c_i \cdot \widetilde{x}_i$.

Zadeh's extension principle in case of linearization. For linear functions, the general Zadeh's extension principle takes the form

$$\mu(y) = \max\left\{ f_\&(\mu_1(x_1), \ldots, \mu_n(x_n)) : a_0 + \sum_{i=1}^{n} c_i \cdot x_i = y \right\}.$$

In particular, the min-based extension principle takes the form

$$\mu(y) = \max\left\{ \min(\mu_1(x_1), \ldots, \mu_n(x_n)) : a_0 + \sum_{i=1}^{n} c_i \cdot x_i = y \right\}.$$

In the linearized case, we can reduce a general problem to several sums of two variables. From the computational viewpoint, linearization has an additional advantage: it enables us to reduce a general problem, with many inputs, to a sequence of problems with only two inputs. To be more precise, we can reduce it to the problem of computing the sum of two variables.

Indeed, our goal is to compute the membership degrees corresponding to the sum

$$y = a_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n.$$

To simplify this problem, let us recall how this expression would be computed on a computer.

- First, we will compute $y_1 \stackrel{\text{def}}{=} c_1 \cdot x_1$, then the first intermediate sum $s_1 = a_0 + y_1$.
- After that, we will compute the second product $y_2 = c_2 \cdot x_2$, and the second intermediate sum $s_2 = s_1 + y_2$.
- Then, we compute the third product $y_3 = c_3 \cdot x_3$, and the third intermediate sum $s_3 = s_2 + y_3$.
- ...
- At the end, we compute $y_n = c_n \cdot x_n$ and $y = s_{n-1} + y_n$.

let us follow the same pattern when computing the membership function $\mu(y)$. First, let us find the membership functions $\nu_i(y_i)$ corresponding to $y_i = c_i \cdot x_i$. For each y_i, there is only one value x_i for which $y_i = c_i \cdot x_i$ – namely, the value $x_i = y_i/c_i$ – so in Zadeh's formula, there is no need to apply "or" or "and", we just have $\nu_i(y_i) = \mu_i(y_i/c_i)$.

Similarly, for $s_1 = a_0 + y_1$, we have $\eta_1(s_1) = \nu_1(s_1 - a_0) = \mu_1((s_1 - a_0)/c_1)$. Now, we can do the following:

- first, based on the membership functions $\eta_1(s_1)$ and $\nu_2(y_2)$, we find the membership function $\eta_2(s_2)$ corresponding to the sum $s_2 = s_1 + y_2$,

- then, based on the membership functions $\eta_2(s_2)$ and $\nu_3(y_3)$, we find the membership function $\eta_3(s_3)$ corresponding to the sum $s_3 = s_2 + y_3$,
- etc.
- until, based on the membership functions $\eta_{n-1}(s_{n-1})$ and $\nu_n(y_n)$, we find the desired membership function $\mu(y)$ corresponding to the sum $y = s_{n-1} + y_n$.

On each step, we need to several times apply Zadeh's extension principle to the sum of two variables.

For the sum of two variables, these formulas have the following simplified form.

Zadeh's extension principle: case of the sum of two variables. For a general "and"-operation $f_\&(a, b)$, when we know the membership functions $\mu_1(x_1)$ and $\mu_2(x_2)$ describing two quantities x_1 and x_2, then, once we have selected x_1, the value x_2 for which $x_1 + x_2 = y$ can be easily described as $x_2 = y - x_1$. Thus, the membership function $\mu(y)$ for the sum $y = x_1 + x_2$ takes the form

$$\mu(y) = \max_{x_1} f_\&(\mu_1(x_1), \mu_2(y - x_1)).$$

In particular, for $f_\&(a, b) = \min(a, b)$, we have

$$\mu(y) = \max_{x_1} \min(\mu_1(x_1), \mu_2(y - x_1)).$$

In the following text, this is what we will consider: algorithms for computing these two formulas.

3 Efficient Fuzzy Data Processing for the min t-Norm: Reminder

Our objective is to come up with efficient algorithm for fuzzy data processing for a general "and"-operation.

To come up with such an algorithm, let us recall how fuzzy data processing is performed in the case of the min t-norm.

What is the input for the algorithm. To describe each of the two input membership functions $\mu_1(x_1)$ and $\mu_2(x_2)$, we need to describe their values at certain number of points $v_0 < v_1 < \ldots < v_{N-1}$, where N is the total number of these points. Usually, these points are equally spaced, i.e., $v_i = v_0 + i \cdot \Delta$ for some $\Delta > 0$. Thus, the inputs consist of $2N$ values $\mu_i(v_j)$ corresponding to $i = 1, 2$ and to $j = 0, 1, \ldots, N - 1$.

First – naive – algorithm for fuzzy data processing. When the values of each of the two variables x_1 and x_2 go from v_0 to $v_{N-1} = v_0 + (N - 1) \cdot \Delta$, their sum $y = x_1 + x_2$ takes values $w_0 \overset{\text{def}}{=} 2v_0$, $w_1 = w_0 + \Delta = 2v_0 + \Delta$, ..., all the way to $w_{2(N-1)} = 2v_{N-1} = 2v_0 + 2 \cdot (N - 1) \cdot \Delta$.

Each value w_k can be represented as the sum $v_i + v_{k-i}$ of possible values of x_1 and x_2 when $0 \le i \le N-1$ and $0 \le k-i \le N-1$, i.e., when

$$\max(k+1-N, 0) \le i \le \min(k, N-1).$$

Thus, we have

$$\mu(w_k) = \max(\min(\mu_1(v_i), \mu_2(v_{k-i})) : \max(k+1-N, 0) \le i \le \min(k, N-1)\}.$$

What is the computational complexity of this naive algorithm. The computational complexity of an algorithm is usually gauged by its number of computational steps – which is roughly proportional to the overall computation time.

In the above case, for each k, we have up to N different values i:

- exactly N for $k = N-1$,
- $N-1$ for $k = N-2$ and $k = N$,
- $N-2$ values i for $k = N-3$ and $k = N+1$,
- etc.

For each k and i, we need one min operation, to compute the minimum

$$\min(\mu_1(v_i), \mu_2(v_{k-i})).$$

Then, we need to find the maximum of all these numbers. So, we need N steps to compute the value w_{N-1}, $N-1$ steps to compute the values w_{N-2} and w_N, etc. Overall, we therefore need

$$N = 2\cdot(N-1) + 2\cdot(N-2) + \ldots + 2\cdot 1 = N + 2\cdot((N-1) + (N-2) + \ldots + 1) =$$

$$N + 2\cdot\frac{N\cdot(N-1)}{2} = N + N\cdot(N-1) = N^2$$

computational steps.

It is possible to compute $\mu(y)$ faster, by using α-cuts. It is known that for the case of the min t-norm, it is possible to compute $\mu(y)$ faster. This possibility comes from considering *α-cuts*

$$\mathbf{x}_i(\alpha) \stackrel{\text{def}}{=} \{x_i : \mu_i(x_i) \ge \alpha\}.$$

Indeed, according to Zadeh's formula, $\mu(y) \ge \alpha$ if and only if there exists values x_1 and x_2 for which $x_1 + x_2 = y$ and $\min(\mu_1(x_1), \mu_2(x_2)) \ge \alpha$, i.e., equivalently, for which $\mu_1(x_1) \ge \alpha$ and $\mu_2(x_2) \ge \alpha$.

Thus, the α-cut for y is equal to the set of all possible values $y = x_1 + x_2$ when x_1 belongs to the α-cut for x_1 and x_2 belongs to the α-cut for x_2:

$$\mathbf{y}(\alpha) = \{x_1 + x_2 : x_1 \in \mathbf{x}_1(\alpha) \,\&\, x_2 \in \mathbf{x}_2(\alpha)\}.$$

In many practical situations, each of the membership functions $\mu_i(x_i)$ increases up to a certain value, then decreases. In such situations, each α-cut is an interval: $\mathbf{x}_i(\alpha) = [\underline{x}_i(\alpha), \overline{x}_i(\alpha)]$. If we know that x_1 belongs to the interval $[\underline{x}_1(\alpha), \overline{x}_1(\alpha)]$ and that x_2 belongs to the interval $[\underline{x}_2(\alpha), \overline{x}_2(\alpha)]$, then possible values of $y = x_1 + x_2$ form the interval

$$[\underline{y}(\alpha), \overline{y}(\alpha)] = [\underline{x}_1(\alpha) + \underline{x}_2(\alpha), \overline{x}_1(\alpha) + \overline{x}_2(\alpha)].$$

(It is worth mentioning that this formula is a particular case of *interval arithmetic*; see, e.g., [2, 5].)

Thus, instead of N^2 elementary operations, we only need to perform twice as many operations as there are possible levels α. When we use $2N$ values $\mu_i(x_i)$, we can have no more than $2N$ different values α, so the computation time is $O(N) \ll N^2$.

This is the reason why α-cuts – and interval computations – are mostly used in fuzzy data processing.

4 Efficient Fuzzy Data Processing Beyond min t-Norm: the Main Result of This Paper

Now that we recalled how fuzzy data processing can be efficiently computed for the min t-norm, let us find out how we can efficiently compute it for the case of a general t-norm. Reminder: we know the membership functions $\mu_1(x_1)$ and $\mu_2(x_2)$, we want to estimate the values

$$\mu(y) = \max_{x_1} f_\&(\mu_1(x_1), \mu_2(y - x_1)).$$

Naive algorithm. Similar to the case of the min t-norm, the naive (straightforward) application of this formula leads to an algorithm that requires N^2 steps: the only difference is that now, we use $f_\&$ instead of min.

Let us show that for t-norms different from min, it is also possible to compute $\mu(y)$ faster.

Reducing to the case of a product t-norm. It is know that Archimedean t-norms, i.e., t-norms of the type $f^{-1}(f(a) \cdot f(b))$ for monotonic $f(x)$, are *universal approximators*, in the sense that for every t-norm and for every $\varepsilon > 0$, there exists an Archimedean t-norm whose value is ε-close to the given one for all possible a and b; see, e.g., [6].

Thus, from the practical viewpoint, we can safely assume that the given t-norm is Archimedean. For the Archimedean t-norm, the above formula takes the form

$$\mu(y) = \max_{x_1} f^{-1}(f(\mu_1(x_1)) \cdot f(\mu_2(y - x_1))).$$

The inverse function f^{-1} is monotonic, thus, the largest values of $f^{-1}(z)$ is attained when z is the largest. So, we have

$$\mu(y) = f^{-1}(\nu(y)),$$

where we denoted

$$\nu(y) = \max_{x_1} f(\mu_1(x_1)) \cdot f(\mu_2(y - x_1)).$$

So, if we denote $\nu_1(x_1) \stackrel{\text{def}}{=} f(\mu_1(x_1))$ and $\nu_2(x_2) \stackrel{\text{def}}{=} f(\mu_2(x_2))$, we arrive at the following problem: given functions $\nu_1(x_1)$ and $\nu_2(x_2)$, compute the function

$$\nu(y) = \max_{x_1}(\nu_1(x_1) \cdot \nu_2(y - x_1)).$$

This is exactly the original problem for the product t-norm. Thus, fuzzy data processing problem for a general t-norm can indeed be reduced to the problem of fuzzy data processing for the product t-norm.

Let us simplify our problem. Multiplication can be simplified if we take logarithm of both sides; then it is reduced to addition. This is why logarithms were invented in the first place – and this is why they were successfully used in the slide rule, the main computational tool for the 19 century and for first half of the 20 century – to make multiplication faster.

The values $\nu_i(x_i)$ are smaller than 1, so their logarithms are negative. To make formulas easier, let us use positive numbers, i.e., let us consider the values $\ell_1(x_1) \stackrel{\text{def}}{=} -\ln(\nu_1(x_1))$, $\ell_2(x_2) \stackrel{\text{def}}{=} -\ln(\nu_2(x_2))$, and $\ell(y) \stackrel{\text{def}}{=} -\ln(\nu(y))$. Since we changed signs, max becomes min, so we get the following formula:

$$\ell(y) = \min_{x_1}(\ell_1(x_1) + \ell_2(y - x_1)).$$

We have thus reduced our problem to a known problem in convex analysis. The above formula is well-known in *convex analysis* [9]. It is known as the *infimal covolution*, or an *epi-sum*, and usually denoted by

$$\ell = \ell_1 \,\Box\, \ell_2.$$

There are efficient algorithms for solving this convex analysis problem. At least for situations when the functions $\ell_i(x_i)$ are convex (and continuous), there is an efficient algorithm for computing the infimal convolution. This algorithm is based on the use of *Legendre-Fechnel transform*

$$\ell^*(s) = \sup_x(s \cdot x - \ell(x)).$$

Specifically, it turns out that the Legender transform of the infimal convolution is equal to the sum of the Legendre transforms:

$$(\ell_1 \,\Box \ell_2)^* = \ell_1^* + \ell_2^*,$$

and that to reconstruct a function form its Legendre transform, it is sufficient to apply the Legendre transform once again:

$$\ell = (\ell^*)^*.$$

Thus, we can compute the infimal convolution as follows:

$$\ell_1 \,\Box\, \ell_2 = (\ell_1^* + \ell_2^*)^*.$$

Similarly, we can compute the infimal composition of several functions

$$\ell_1 \,\Box\, \ldots \,\Box\, \ell_n = \min\{\ell_1(y_1) + \ldots + \ell_n(y_n) : y_1 + \ldots + y_n = y\},$$

as

$$\ell_1 \,\Box\, \ldots \,\Box\, \ell_n = (\ell_1^* + \ldots + \ell_n^*)^*.$$

There exists a fast (linear-time $O(N)$) algorithm for computing the Legendre transform; see, e.g., [4]. So, by using this algorithm, we can compute the results of fuzzy data processing in linear time.

Let us summarize the resulting algorithm.

5 Resulting Linear Time Algorithm for Fuzzy Data Processing Beyond $\min$ t-Norm

What is given and what we want to compute: reminder. We are given:

- a function $f(x_1, \ldots, x_n)$;
- n membership functions $\mu_1(x_1)$, ..., $\mu_n(x_n)$; and
- an "and"-operation $f_\&(a, b)$.

We want to compute a new membership function

$$\mu(y) = \max\{f_\&(\mu_1(x_1), \ldots, \mu_n(x_n)) : f(x_1, \ldots, x_n) = y\}.$$

Algorithm.

- First, we represent the given "and"-operation in the Acrhiemdean form $f_\&(a, b) = f^{-1}(f(a) \cdot f(b))$ for an appropriate monotonic function $f(x)$.

  In the following text, we will assume that we have algorithms for computing both the function $f(x)$ and the inverse function $f^{-1}(x)$.

- Then, for each i, we find the value $\widetilde{x}_i$ for which $\mu_i(x_i)$ attains its largest possible value.

  For normalized membership functions, this value is $\mu_i(\widetilde{x}_i) = 1$.

- We then compute the values $c_0 = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $c_i = \dfrac{\partial f}{\partial x_i}_{|x_i = \widetilde{x}_i}$, and

$$a_0 = c_0 - \sum_{i=1}^{n} c_i \cdot \widetilde{x}_i.$$

In the following text, we will then use a linear approximation

$$f(x_1, \dots, x_n) = a_0 + \sum_{i=1}^{n} c_i \cdot x_i.$$

- After that, we compute the membership functions

$$\nu_1(s_1) = u_1((s_1 - a_0)/c_1)$$

  and $\nu_i(y_i) = \mu_i(y_i/c_i)$ for $i > 2$.

  In terms of the variables $s_1 = a_0 + c_1 \cdot x_1$ and $y_i = c_i \cdot x_i$, the desired quantity y has the form $y = s_1 + y_2 + \dots + y_n$.

- We compute the minus logarithms of the resulting functions:

$$\ell_i(y_i) = -\ln(\nu_i(y_i)).$$

- For each i, we then use the Fast Legendre Transform algorithm from [4] to compute ℓ_i^*.
- Then, we add all these Legendre transforms and apply the Fast Legendre Transform once again to compute the function

$$\ell = (\ell_1^* + \dots + \ell_n^*)^*.$$

- This function $\ell(y)$ is equal to $\ell(y) = -\ln(\nu(y))$, so we can reconstruct $\nu(y)$ as

$$\nu(y) = \exp(-\ell(y)).$$

- Finally, we can compute the desired membership function $\mu(u)$ as

$$\mu(y) = f^{-1}(\nu(y)).$$

6 Conclusions

To process fuzzy data, we need to use Zadeh's extension principle. In principle, this principle can be used for any t-norm. However, usually, it is only used for the min t-norm, since only for this t-norm, an efficient (linear-time) algorithm for fuzzy data processing was known.

Restricting oneselves to min t-norm is not always a good idea, since it is known that in many practical situations, other t-norms are more adequate in describing expert's reasoning. In this paper, we show that similar efficient linear-time algorithms can be designed for an arbitrary t-norm. Thus, it become possible to use a t-norm that most adequately describes expert reasoning in this particular domain – and still keep fuzzy data processing algorithm efficient.

Acknowledgments

This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

We also acknowledge the partial support of the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand.

References

[1] B. G. Buchanan and E. H. Shortliffe, *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, Massachusetts, 1984.

[2] L. Jaulin, M. Kiefer, O. Dicrit, and E. Walter, *Applied Interval Analysis*, Springer, London, 2001.

[3] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[4] Y. Lucet, "Faster than the Fast Legendre Transform, the Linear-time Legendre Transform", *Numerical Algorithms*, 1997, Vol. 16, pp. 171–185.

[5] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM, Philadelphia, 2009.

[6] H. T. Nguyen, V. Kreinovich, and P. Wojciechowski, "Strict Archimedean t-Norms and t-Conorms as Universal Approximators", *International Journal of Approximate Reasoning*, 1998, Vol. 18, Nos. 3–4, pp. 239–249.

[7] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.

[8] S. G. Rabinovich, *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, Berlin, 2005.

[9] R. T. Rockafeller, *Convex Snslysis*, Princeton University Press, Princeton, New Jersey, 1997.

[10] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.