In Its Usual Formulation, Fuzzy Computation Is, In General, NP-Hard, But a More Realistic Formulation Can Make It Feasible

Martine Ceberio, Olga Kosheleva,
Vladik Kreinovich, and Luc Longpré
University of Texas at El Paso
El Paso TX 79968, USA
mceberio@utep.edu, olgak@utep.edu
vladik@utep.edu, longpre@utep.edu

1. Outline

- Often, a quantity y depends, in a known way, on quantities $x_1, \ldots, x_n$.
- Zadeh's extension principle leads to useful formulas for computing the membership function for y based on membership functions for x_i.
- However, the challenge is that the corresponding computational problem is NP-hard.
- We present a realistic modification of Zadeh's extension principle.
- For this modification, there is a feasible algorithm for solving the corresponding fuzzy computation problem.

2. Need for Computations

- The main objectives of science and engineering are:
  - to describe the world,
  - to predict what will happen in the future, and,
  - if necessary, to come up with recommendation of what to do to make the future state of the world better.
- The physical world is usually described by the values of the corresponding physical quantities.
- Thus, to describe the current state of the world, we need to describe the numerical values of all these quantities.
- Some of these values we can direct measure or estimate.
- We can directly measure the width of a room.
- By touching a baby's forehead, we can directly estimate the baby's body temperature, etc.
- However, there are many other quantities which are difficult to measure or estimate directly.

3. Need for Computations (cont-d)

- For example, it is not easy to directly measure or estimate the distance to a faraway star, or the temperature inside the car engine.
- This impossibility is even more evident if we are interested in the future values of the quantities of interest.
- In such cases, natural idea is to estimate this value indirectly:
  - we find easier-to-or-estimate quantities $x_1, \ldots, x_n$ related to y by a known dependence $y = f(x)$, where $x \stackrel{\text{def}}{=} (x_1, \ldots, x_n)$;
  - then, we measure or estimate these auxiliary quantities x_i;
  - finally, we use the resulting estimates $\widetilde{x}_i$ to compute the estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for the desired quantity y.
- For example, to predict the temperature y in El Paso in a week, we can:
  - measure the values $x_1, \ldots, x_n$ describing the temperature, humidity, and wind speed measurements in a wide area, and then
  - use the algorithm $y = f(x_1, \ldots, x_n)$ for solving the corresponding partial differential equations to predict y.
- Such estimations are the main reason why computations are needed.

4. Traditional Formulas for Fuzzy Computing: Zadeh's Extension Principle

- How can we compute $\mu(y)$ based on $\mu_i(x_i)$?
- Y is a possible value of $y = f(x_1 \ldots, x_n)$ if $Y = f(X_1, \ldots, X_n)$ for some possible values X_i of x_i.
- In other words, Y is possible if:
  - either X_1 is a possible value of x_1 *and* X_2 is a possible value of x_2, *and* $\ldots$, for some tuple $(X_1, \ldots, X_n)$ for which $Y = f(X_1, \ldots, X_n)$,
  - *or* X_1' is a possible value of x_1 *and* X_2' is a possible value of x_2, *and* $\ldots$, for some tuple $(X_1', \ldots, X_n')$ for which $Y = f(X_1', \ldots, X_n')$,
  - or the same us true for other values $X_1'', \ldots, X_n''$.
- For each i and for each value X_i, we know the degree $\mu_i(X_i)$ to which X_i is a possible value of x_i.
- So, if we interpret "and" as min as "or" as max, we get

$$\mu(Y) = \max_{X_1,\ldots,X_n : Y = f(X_1,\ldots,X_n)} \min \{\mu_1(X_1), \mu_2(X_2), \ldots\}.$$

- This formula is known as *Zadeh's extension principle.*

5. Zadeh's Extension Principle Is NP-Hard

- Zadeh's extension principle can be naturally expressed in terms of α-cuts $\mathbf{y}(\alpha) \stackrel{\text{def}}{=} \{y : \mu(y) \geq \alpha\}$ and $\mathbf{x}_i(\alpha) \stackrel{\text{def}}{=} \{x_i : \mu_i(x_i) \geq \alpha\}$:
$$\mathbf{y}(\alpha) = \{f(x_1, \ldots, x_n) : x_i \in \mathbf{x}_i(\alpha) \text{ for all } i\}\,.$$
- It is known that even when the sets $\mathbf{x}_i(\alpha)$ are intervals, computing the range is NP-hard for quadratic functions $f(x_1, \ldots, x_n)$.
- So, unless P = NP, no general feasible algorithm is possible for performing fuzzy computations.
- For any fuzzy computations algorithm, time complexity grows very fast with the number of variables n.

6. Related Work

- The relation between fuzziness and NP-hardness is well known and well exploited.
- Many authors have used fuzzy techniques to provide efficient algorithms for solving particular cases of NP-hard problems.
- This paper is different:
  - instead of using fuzzy techniques to solve NP-hard problems,
  - it shows how to modify a fuzzy computation problem so that it stops being NP-hard.

7. Our Main Idea

- The usual derivation of Zadeh's extension principle considers *all* possible tuples $(X_1, \ldots, X_n)$ for which $f(X_1, \ldots, X_n) = Y$.
- Similarly, in the formulas for the α-cut, we consider *all* possible tuples $(x_1, \ldots, x_n)$ for which $\mu_i(x_i) \geq \alpha$ for every i.
- In both cases, we took "all" literally: all means all, one exception makes a statement about all the tuples false.
- From the mathematical viewpoint, this is a reasonable idea.
- But let us take into account that we are not proving mathematical theorems.
- We are trying to formalize common sense, we are trying to formalize expert reasoning.
- In our usual reasoning, "all" does not mean mathematically all.
- It usually means "almost all", meaning everyone except a small fraction of the original population.

8. Our Main Idea (cont-d)

- When a patriotic journalist says all the citizens support their government, he usually mentions a new dissenters.
- When we say that all pigeons can fly, we understand very well that there may be a wounded or deformed pigeon, but that most pigeons can fly.
- A classical AI example is a phrase "all birds fly".
- This phrase has known exceptions, such as penguins, but the vast majority of the birds indeed can fly.
- Let us see how the above definitions of fuzzy computing will change if we use a commonsense meaning of "all".

9. Towards a New Formalization of Fuzzy Computing

- Let us define $\overline{y}$ as the maximum of "almost all" values.
- Let us fix the exact proportion $\delta > 0$ of values that we can ignore.
- Then, we are looking for a value $\overline{y}$ for which

$$\frac{|\{x : x_1 \in \mathbf{x}_1 \,\&\, \ldots \,\&\, x_n \in \mathbf{x}_n \,\&\, f(x_1, \ldots, x_n) \le \overline{y}\}|}{|\{x : x_1 \in \mathbf{x}_1 \,\&\, \ldots \,\&\, x_n \in \mathbf{x}_n\}|} 1 - \delta.$$

- Here $|S|$ denotes the multi-D volume of a set S:
  - width of an interval,
  - area of a planar (2-D) set,
  - volume of a 3-D set, etc.
- When δ tends to 0, the corresponding value tends to the maximum of the function $f(x_1, \ldots, x_n)$ on the box $\mathbf{x} \stackrel{\text{def}}{=} \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$.
- Thus, for small δ, the above-defined value is very close to this maximum.
- Similarly, $\underline{y}$ is the value for which

$$\frac{|\{x : x \in \mathbf{x} \,\&\, f(x) \ge \underline{y}\}|}{|\{x : x \in \mathbf{x}\}|} = 1 - \delta.$$

10. Towards a New Formalization (cont-d)

- Intuitively, since we are considering the *fuzzy* case, it makes no sense to fix one *exact* value δ.
- It is more appropriate to assume that this value is also given with some uncertainty.
- Let us assume that we know the interval $[\underline{\delta}, \overline{\delta}\,]$, with $\underline{\delta} < \overline{\delta}$, that contains the actual (unknown) value δ.
- Thus, e.g., for $\overline{y}$ we get the double inequality:

$$1 - \overline{\delta} \le \frac{|\{x : x \in \mathbf{x} \,\&\, f(x) \le \overline{y}\}|}{|\{x : x \in \mathbf{x}\}|} \le 1 - \underline{\delta}.$$

11. What Does It Mean to Compute $\underline{y}$ and $\overline{y}$?

- We relaxed the requirement on the endpoints $\underline{y}$ and $\overline{y}$.
- It makes sense to also relax the usual requirement on the algorithm: that it always computes the desired value.
- From the practical viewpoint, it makes sense to consider algorithms that provide an answer with a probability $1 - p_0$, for some small $p_0 \ll 1$.
- Indeed, even the computer hardware is not 100% reliable, once in a while computers break down.
- From this viewpoint, it is perfectly OK if the algorithm also sometimes does not produce the desired result.
- As long as the probability for this is much smaller than the probability of a hardware fault, we are OK.

12. Resulting Definition

- Let $\varepsilon > 0$ be a rational number.
- We say that a function $f(x_1, \ldots, x_n)$ is ε-*feasible* if there exists a feasible algorithm that:
  - given rational values $x_1, \ldots, x_n$,
  - produces a rational number which is ε-close to $f(x_1, \ldots, x_n)$.
- Let $\varepsilon > 0$, $0 < \underline{\delta} < \overline{\delta}$, and $p_0 > 0$ be rational numbers.
- By *realistic fuzzy computations*, we mean the following problem:
- GIVEN: rational numbers $\underline{x}_1, \overline{x}_1, \ldots, \underline{x}_n, \overline{x}_n$, and an ε-feasible function $f(x_1, \ldots, x_n)$ with rational coefficients,
- COMPUTE, with probability $\geq 1 - p_0$, rational numbers $\underline{r}$ and $\overline{r}$ which are ε-close to, correspondingly, values $\underline{y}$ and $\overline{y}$ for which

$$1 - \overline{\delta} \leq \frac{|\{x : x \in \mathbf{x} \,\&\, f(x) \geq \underline{y}\}|}{|\{x : x \in \mathbf{x}\}|} \leq 1 - \underline{\delta} \text{ and}$$

$$1 - \overline{\delta} \leq \frac{|\{x : x \in \mathbf{x} \,\&\, f(x) \leq \overline{y}\}|}{|\{x : x \in \mathbf{x}\}|} \leq 1 - \underline{\delta}.$$

13. Main Result

- For each tuple $(\varepsilon, \underline{\delta}, \overline{\delta}, p_0)$, there exists a feasible algorithm that solves the corresponding realistic fuzzy computations problem.
- The desired algorithm uses the standard random number generator that generates numbers ξ uniformly distributed on the interval $[0, 1]$.
- For each interval $[\underline{x}_i, \overline{x}_i]$, we can compute the value $x_i = \underline{x}_i + \xi \cdot (\overline{x}_i - \underline{x}_i)$ uniformly distributed on the interval $[\underline{x}_i, \overline{x}_i]$.
- If we repeat this procedure n times, for n intervals $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, then we get a tuple $(x_1, \ldots, x_n)$ which is uniformly distributed on the box

$$\mathbf{x}_1 \times \ldots \times \mathbf{x}_n.$$

- Now, we can formulate the resulting algorithm.
- First, we select an appropriate natural number N, and compute $\widetilde{\delta} \stackrel{\text{def}}{=} \dfrac{\underline{\delta} + \overline{\delta}}{2}$, $\Delta \stackrel{\text{def}}{=} \dfrac{\overline{\delta} - \underline{\delta}}{2}$, and $v = \lfloor N \cdot \widetilde{\delta} \rfloor$.
- Then, N times we use the above procedure for generating tuples uniformly distributed on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, and get N tuples $x^{(k)}$.

14. Main Result (cont-d)

- We apply the given feasible algorithm f to each of these tuples, generating N values $y^{(k)}$ for which $\left|y^{(k)} - f\left(x^{(k)}\right)\right| \le \varepsilon$.
- We sort $y^{(k)}$ into an increasing sequence $y_{(1)} \le \ldots \le y_{(N)}$.
- Finally, we take $\underline{r} = y_{(v)}$ and $\overline{r} = y_{(N-v)}$.
- One can show that for an appropriate N, this algorithm solves the corresponding realistic fuzzy computation problem.
- Our preliminary results show that this algorithm is not just theoretically feasible: it indeed produces the desired result in reasonable time.
- We hope that practitioners will apply our algorithm to practical problems and thus, test its efficiency.
- Shall we worry about the use of Monte-Carlo techniques in a paper about fuzzy computation?
- Many papers on fuzzy computations emphasize that their results are faster to compute than more traditional Monte-Carlo-based techniques.

15. Main Result (cont-d)

- However, there is no contradiction; indeed:
  - the computation time of usual fuzzy computation algorithms grows with n, while
  - the number of iterations N needs for a Monte-Carlo algorithm does not depend on n at all.
- So, for large n, Monte-Carlo-type methods become more efficient.
- When the number of inputs n is reasonably small, the usual methods are much faster – which is exactly what many papers have claimed.
- Similar ideas can be applied to come up with feasible algorithms for solving more complex problems, such as minimax or maximin:

$$\min_{x \in X} \max_{y \in Y} f(x, y) \text{ or } \max_{x \in X} \min_{y \in Y} f(x, y).$$

- This is important, e.g., in game theory.