4-2020

Which Algorithms Are Feasible and Which Are Not: Fuzzy Techniques Can Help in Formalizing the Notion of Feasibility

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-20-29

Which Algorithms Are Feasible and Which Are Not: Fuzzy Techniques Can Help in Formalizing the Notion of Feasibility

Olga Kosheleva and Vladik Kreinovich

Abstract Some algorithms are practically feasible, in the sense that for all inputs of reasonable length they provide the result in reasonable time. Other algorithms are not practically feasible, in the sense that they may work well for small-size inputs, but for slightly larger – but still reasonable-size – inputs, the computation time becomes astronomical (and not practically possible). How can we describe practical feasibility in precise terms? The usual formalization of the notion of feasibility states that an algorithm is feasible if its computation time is bounded by a polynomial of the size of the input. In most cases, this definition works well, but sometimes, it does not: e.g., according to this definition, every algorithm requiring a constant number of computational steps is feasible, even when this number of steps is larger than the number of particles in the Universe. In this paper, we show that by using fuzzy logic, we can naturally come up with a more adequate description of practical feasibility.

1 Formulation of the Problem

Some algorithm are feasible and some are not. Computer scientists have invented many different algorithms. Some of these algorithm are practically feasible, in the sense that for inputs of reasonable size, they require reasonable (and practically implementable) time. Examples of such algorithms include different algorithms for search, for sorting, for solving systems of linear equations, etc.; see, e.g., [2].

On the other hand, there are algorithms which always produce the correct results but which, in practice, only work for small size inputs – otherwise, they require an unrealistic amount of computation time. A good example is an exhaustive search algorithm for solving the propositional satisfiability problem – given a propositional

Olga Kosheleva and Vladik Kreinovich
University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA, e-mail: olgak@utep.edu, vladik@utep.edu

formula (i.e., an expression obtained from Boolean (yes-no) variables $v_1, \ldots, v_n$ by using "and", "or", and "not"), find the values of these variables that make the formula true. In principle, we can solve this problem by trying all 2^n possible tuples of values $(v_1, \ldots, v_n)$ – each variable has two possible values (true of false), so the tuple has 2^n possible values.

- It works for $n = 10$, when we need $2^{10} \approx 10^3$ computational steps.
- It works for $n = 20$, when we need $2^{20} \approx 10^6$ steps.
- It works for $n = 30$, when we need $2^{30} \approx 10^9$ computational steps, one second or less on a usual GigaHerz computer.

However, already for a very reasonable size input $n = 300$, we will need $2^{300} \approx 10^{100}$ computational steps – which would require time which is much much longer than the lifetime of the Universe. So this algorithm is clearly not practically feasible.

It is desirable to have a precise definition of feasibility. It would be nice to know which algorithm is practically feasible and which is not. It is not easy to make such a conclusion based on the above description of practical feasibility, since this description uses imprecise works like "reasonable". To make the corresponding conclusion, it is desirable to have a precise definition of what is feasible.

How is the notion of feasibility described now. The existing formal definition of feasibility is based on the following fact:

- for the vast majority of practically feasible algorithms – including search, sorting, solving systems of linear equations – the worst-case computation time $t(n)$ on inputs of size n is bounded by some polynomial of n, while
- for the vast majority of not practically feasible algorithms – like the above-described exhaustive search algorithm – the worst-case computation time is exponential – or at least grows faster than any polynomial.

Because of this fact, formally, an algorithm is called *feasible* if its worst-case computation time $t(n)$ is bounded by some polynomial – i.e., if there exists a polynomial $P(n)$ for which $t(n) \le P(n)$ for all n; see, e.g., [4, 8].

The current formal definition is not fully adequate. In many cases, the above formal definition correctly describes what is feasible and what is not feasible. However, there are cases when this definition does not adequately describe practical feasibility. Let us give two examples:

- when $t(n) = 10^{100} \cdot n$, this expression is a polynomial – so it is feasible according to the current formal definition – but it is clearly *not* practically feasible, since even for inputs of length 1, this algorithm requires impossible 10^{100} steps to finish; similar arguments can be given if $t(n)$ is a large constant – e.g., if $t(n) = 10^{100}$ for all input sizes n;
- on the other hand, when $t(n) = \lceil \exp(10^{-20} \cdot n) \rceil$, then, strictly speaking, it is an exponential function, so it grows faster than any polynomial (and is, thus, not feasible in the sense of the formal definition), but even when we input the whole body of current knowledge, with $n = 10^{18}$, this algorithm will work really fast – in $\lceil \exp(10^{-20} \cdot 10^{18}) \rceil = \lceil \exp(0.01) \rceil = 2$ steps.

So, we arrive at a natural question.

A natural question, and what we do in this paper. Can we come up with an alternative precise definition of feasibility that would be more adequate? In this paper, we show that fuzzy techniques (see, e.g., [1, 3, 5, 6, 7, 9]) can help in providing such a definition.

2 Analysis of the Problem and Possible Solution

Natural idea: using fuzzy techniques. The informal description of practical feasibility uses the natural-language word "reasonable". Like many other natural-language words – like "small", "large", etc. – this word is not precise. Different people may disagree on what is reasonable, and for large but not too large sizes n, even a single person can be unsure whether this size is reasonable or not.

It is precisely to deal with such imprecise ("fuzzy") words from natural language that Lotfi Zadeh invented fuzzy techniques. So, a natural idea is to use fuzzy techniques to formalize the notion of practical feasibility.

Let us apply fuzzy techniques. To use fuzzy techniques, let us first re-formulate the above description of practical feasibility in more precise terms. Practical feasibility means that for all possible length n, if n is reasonable, then $t(n)$ should be reasonable too. If we denote "n is reasonable" by $r(n)$, then the definition of practical feasibility takes the following form $\forall n\,(r(n) \to r(t(n)))$, or, equivalently,

$$(r(1) \to r(t(1))) \,\&\, (r(2) \to r(t(2))) \,\&\, \ldots \tag{1}$$

In fuzzy logic, our degree of confidence in each statement S is described by a number from the interval $[0,1]$:

- the value 1 means that we are absolutely confident that the statement S is true;
- the value 0 means that we are absolutely confident that the statement S is false; and
- values between 0 and 1 indicate intermediate situations, when we are confident only to some extent.

For each imprecise property like $r(n)$, we can describe, for each n, the degree $R(n)$ that this property is true (i.e., in our case, that n is reasonable). The mapping that assigns this degree to each n is known as the *membership function* describing the corresponding notion.

Clearly, if the value n is reasonable, then all smaller values are reasonable as well. Thus, the degree $R(n)$ should be non-strictly decreasing, from $R(1) = 1$ to $R(n) \to 0$ as n increases.

To come up with estimates of composite statements – obtained by using logical connectives like "and" and "if ... then" from the elementary statements – we can use fuzzy analogues of these connectives, i.e., appropriate extensions of the usual logi-

cal connectives form the two-valued set $\{0,1\} = \{\text{false}, \text{true}\}$ to the whole interval $[0,1]$.

The simplest possible "and"-operation is $\min(a,b)$, the simplest possible "or"-operation is $\max(a,b)$, and the simplest possible negation operation is $1-a$. Implication $A \to B$ is, in classical logic, equivalent to $B \vee \neg A$. Thus, if we know the truth values a and b of (= degrees of confidence in) statement A and B, then the truth value of the implication $A \to B$ can be estimated as $\max(b, 1-a)$. Thus, the truth value of the formula (2) – i.e., the degree $D(t)$ to which an algorithm with worst-case time complexity $t(n)$ is practically feasible – takes the following form:

$$D(t) = \min(\max(R(t(1)), 1-R(1)), \max(R(t(2)), 1-R(2)), \ldots) =$$

$$\min_n \max(R(t(n)), 1-R(n)). \tag{2}$$

If we use a general "and"-operation $f_{\&}(a,b)$ and a general implication operation $f_{\&}(a,b)$, we get the following formula:

$$D(t) = f_{\&}(f_{\to}(R(1), R(t(1))), f_{\to}(R(2), R(t(2))), \ldots) \tag{3}$$

This is our precise definition of practical feasibility.

The proposed new precise definition of practical feasibility is indeed more adequate than the existing one. Let us show that already for the simplest possible operations $f_{\&}(a,b) = \min(a,b)$ and $f_{\to}(a,b) = \max(b, 1-a)$, the above definition is more adequate that the existing formal definition.

Indeed, for example, according to the formal definition, any function with constant time $t(n) = t = \text{const}$ is feasible. What will happen is we use our definition (2) – or, to be precise, its simplest-case version (1)? When n increases, the value $R(n)$ decreases, thus the value $1-R(n)$ increases and the value

$$\max(R(t(n)), 1-R(n)) = \max(R(t), 1-R(n))$$

also increases. So, the minimum $D(t)$ is attained when the size n is the smallest, i.e., when $n = 1$:

$$D(t) = \max(R(t), 1-R(1)).$$

When the constant value t is small, this degree is reasonable and the degree $D(t)$ that this computation time corresponds to a practically feasible algorithm is also reasonable. However, as the constant t increases, the value $R(t)$ tends to 0 and thus, $D(t)$ tends to a very small (practically 0) degree of confident $1-R(1)$ that 1 is not feasible – i.e., as desired, such an algorithm stops being feasible for large t. Actually, here $D(t) \le R(t)$, so if the constant t is not reasonable, the corresponding time complexity is not practically feasible.

Similarly, for a function like $t(n) = \exp(10^{-20} \cdot n)$, the value $R(t(n))$ becomes very small for large n – but for large n, $R(n)$ is also close to 0 and thus, $1-R(n)$ is close to 1 and hence, the maximum $\max(R(t(n)), 1-R(n)) \ge 1-R(n)$ is also close to 1. Thus, the fact that the value $R(t(n))$ is small for such huge n does not

affect the minimum $D(t)$, and the degree of confidence that this computation time is practically feasible remains high.

How to actually compute the newly defined degree of feasibility: towards an algorithm. OK, the definition is reasonable, but how can we actually compute the corresponding degree (2)? Even in its simplest form (1), it is defined as the minimum of infinitely many terms!

It turns out that to come up with the degree $D(t)$, there is no need to actually compute all these infinitely many terms. Indeed, we can use the fact that:

- the function $R(n)$ is decreasing and tending to 0,
- thus $1-R(n)$ is increasing and tending to 1,
- while $R(t(n))$ is decreasing and tending to 0.

So, for large n, we thus have $R(t(n)) \le 1-R(n)$.

If this inequality holds for some n, then for $n \ge n'$, due to the above-described monotonicity, we have

$$R(t(n')) \le R(t(n)) \le 1-R(n) \le 1-R(n')$$

thus $R(t(n')) \le 1-R(n')$. So, if this inequality holds for some n, it holds for all larger values n as well. Hence, there exists the smallest value n_0 for which this inequality is true.

For all values $n \ge n_0$, we have

$$\max(R(t(n)), 1-R(n)) = 1-R(n).$$

This term increases with n, thus the smallest possible value of this term is attained when n is the smallest, i.e., when $n = n_0$. For this value n, we have

$$\max(R(t(n_0)), 1-R(n_0)) = 1-R(n_0).$$

For values $n < n_0$, we have $R(t(n)) > 1-R(n)$ and thus,

$$\max(R(t(n)), 1-R(n)) = R(t(n)).$$

This term decreases with n, thus the smallest possible value of this term is attained when n is the largest, i.e., when $n = n_0 - 1$. For this value n, we have

$$\max(R(t(n_0-1)), 1-R(n_0-1)) = R(t(n_0-1)).$$

Thus, to find the smallest possible value of the maximum-expression

$$\max(R(t(n)), 1-R(n)),$$

there is no need to consider all infinitely many values of this expression corresponding to all possible natural numbers n: it is sufficient to consider only two values of this expression, corresponding to $n = n_0$ and to $n = n_0 - 1$. So, we arrive at the following algorithm.

How to actually compute the newly defined degree of feasibility: algorithm. Find the first value n_0 for which $R(t(n)) \le 1 - R(n)$. This value can be found, e.g., by bisection (see, e.g., [2]). Then, for $n_0 > 1$, we have

$$D(t) = \min(R(t(n_0 - 1)), 1 - R(n_0)).$$

Comment. For $n_0 = 1$, we similarly get $D(t) = 1 - R(1)$.

Acknowledgments

This work was supported in part by the US National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence).

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. Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2009.
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. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.
6. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
7. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
8. C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Massachusetts, 1994.
9. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.