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

Olga Kosheleva and Vladik Kreinovich

University of Texas at El Paso
500 W. University
El Paso, Texas 79968, USA
olgak@utep.edu, vladik@utep.edu

1. Some Algorithm Are Feasible

- 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 include algorithms for search, for sorting, for solving systems of linear equations, etc.

2. Some Algorithm Are Not Feasible

- 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 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 $(v_1, \ldots, v_n)$.

3. Some Algorithm Are Not Feasible (cont-d)

- 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, < 1 sec 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.
- This would require time which is much much longer than the lifetime of the Universe.
- So this algorithm is clearly not practically feasible.

4. 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.
- Indeed, this description uses imprecise works like "reasonable".
- To make the corresponding conclusion, it is desirable to have a precise definition of what is feasible.

5. How Is the Notion of Feasibility Described Now

- The existing formal definition of feasibility is based on the following facts.
- For most 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.
- For most not practically feasible algorithms – like the exhaustive search algorithm:
  - the worst-case computation time is exponential
  - or at least grows faster than any polynomial.

6. Current Notion of Feasibility (cont-d)

- Because of this fact, formally:
  - an algorithm is called *feasible*
  - if its worst-case computation time $t(n)$ is bounded by some polynomial.
- So, it is feasible if there exists a polynomial $P(n)$ for which $t(n) \le P(n)$ for all n.

7. 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.
- When $t(n) = 10^{100} \cdot n$, this expression is a polynomial – so it is feasible according to the formal definition.
- However, it is clearly *not* practically feasible.
- Indeed, 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.

8. Not Fully Adequate (cont-d)

- On the other hand, let's consider

$$t(n) = \lceil \exp(10^{-20} \cdot n) \rceil.$$

- It is an exponential function, so it grows faster than any polynomial.
- It is, thus, not feasible in the sense of the formal definition.
- However:
  - even when we input the whole body of current knowledge, with $n = 10^{18}$,
  - this algorithm will work really fast – in

$$\lceil \exp(10^{-20} \cdot 10^{18}) \rceil = \lceil \exp(0.01) \rceil = 2 \text{ steps.}$$

- So, we arrive at a natural question.

9. A Natural Question, and What We Do in This Talk

- Can we come up with an alternative precise definition of feasibility that would be more adequate?
- In this talk, we show that fuzzy techniques can help in providing such a definition.

10. 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.
- For large but not too large sizes n, even a single person can be unsure whether this size is reasonable or not.
- It is to deal with such imprecise ("fuzzy") words that Lotfi Zadeh invented fuzzy techniques.
- So, a natural idea is to use fuzzy techniques to formalize the notion of practical feasibility.

11. Let Us Apply 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.
- Let us 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,

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

12. Let Us Apply Fuzzy Techniques (cont-d)

- In fuzzy logic, our degree of confidence in each statement S is described by a number from $[0, 1]$:
  - 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.
- In our case, $R(n)$ is the degree that n is reasonable.
- The mapping that assigns this degree to each n is known as the *membership function*.

13. Let Us Apply Fuzzy Techniques (cont-d)

- 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.
- We need to come up with estimates of composite statements – obtained by using logical connectives like "and".
- For this, we can use appropriate extensions of the usual logical connectives:
  - from 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 $\max(a, b)$.
- The simplest possible negation operation is $1 - a$.

14. Let Us Apply Fuzzy Techniques (cont-d)

- 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 $\max(b, 1-a)$.
- So, the degree $D(t)$ to which an algorithm with worst-case time complexity $t(n)$ is practically feasible – is:

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

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

$$D(t) = f_\&(f_\to(R(1), R(t(1))), f_\to(R(2), R(t(2))), \ldots)$$

- This is our precise definition of practical feasibility.

15. The New Definition Is More Adequate

- We will show it for the simplest possible operations $f_{\&}(a, b) = \min(a, b)$ and $f_{\to}(a, b) = \max(b, 1 - a)$.
- 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?
- When n increases, the value $R(n)$ decreases.
- So, $1 - R(n)$ increases.
- Thus, $\max(R(t(n)), 1 - R(n)) = \max(R(t), 1 - R(n))$ 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.

16. The Definition Is More Adequate (cont-d)

- However, as the constant t increases, the value $R(t)$ tends to 0.
- Thus, $D(t)$ tends to a very small (practically 0) degree of confident $1 - R(1)$ that 1 is not feasible.
- Thus, as desired, such an algorithm stops being feasible for large t.
- Actually, here $D(t) \leq R(t)$.
- 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.
- However, for large n, $R(n)$ is also close to 0 and thus, $1 - R(n)$ is close to 1.

17. The Definition Is More Adequate (cont-d)

- Hence, $\max(R(t(n)), 1 - R(n)) \geq 1 - R(n)$ 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 $D(t)$.
- Thus, the degree of confidence that this computation time is practically feasible remains high.

18. How to Compute the Degree of Feasibility

- OK, the definition is reasonable, but how can we actually compute the corresponding degree?
- Even in its simplest form, it is defined as the minimum of infinitely many terms!
- It turns out that to compute $D(t)$, 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 $R(t(n)) \leq 1 - R(n)$.

19. Computing the Degree of Feasibility (cont-d)

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

$$R(t(n')) \le R(t(n)) \le 1 - R(n) \le 1 - R(n') \text{ thus}$$

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

- 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 $n = n_0$.

20. Computing the Degree of Feasibility (cont-d)

- 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

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

21. Computing the Degree of Feasibility (cont-d)

- Thus:
  - to find the smallest possible value of the maximum-expression,
  - it is sufficient to consider only two values of this expression: $n = n_0$ and $n = n_0 - 1$.
- So, we arrive at the following algorithm.

22. Resulting 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.
- Then, for $n_0 > 1$, we have

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

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

23. 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),
- HRD-1242122 (Cyber-ShARE Center of Excellence).