What is Practically Feasible: A Feasible Approach

J. Contreras[1], N. Aun[1], J. Perez[1], S. Ayala[1],
I. Hernandez[1], M. Iglesias[1], D. Obrien[1],
O. Kosheleva[2], and V. Kreinovich[1]
[1]Department of Computer Science
[2]Department of Teacher Education
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA

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, and provide a numerical example to demonstrate the algorithm.

1 Formulation of the Problem

What is practically feasible: a challenging question. Some algorithms are practically feasible; that is, they require reasonable time for inputs of reasonable length. Other algorithms are not practically feasible. How can we describe what is practically feasible in precise terms?

The best known approximation to practical feasibility is the general notion of feasibility in theoretical computer science. In theoretical computer science, the feasibility of an algorithm is defined as follows:

Definition 1.1. Formal Feasibility: An algorithm is feasible if there exists a polynomial P(n) such that for each input x of length n, the computation time t(x) is bounded by P(n).

However, there are practically feasible algorithms that are not captured as feasible using this definition. Conversely, there are algorithms that are captured as feasible using this formal definition of feasibility that are clearly not feasible. We will refer to these algorithms as practically but not formally feasible, and formally but not practically feasible, respectively. The following examples illustrate:

- **Practically but not formally feasible algorithm:** An algorithm whose worst-case complexity is bounded by $\mathrm{t}(n) = \exp(10^{-10}\cdot \mathrm{n})$. Formally, this algorithm is not feasible since it grows faster than a polynomial. However, even if we input an unreasonably large n, say, 10^{10}, we get $\exp(10^{-10}\cdot 10^{10}) = \lceil exp(1) \rceil = 3$ steps. This is clearly a feasible algorithm.

- **Formally but not practically feasible algorithm:** An algorithm whose worse-case complexity is bounded by $\mathrm{t}(n) = 10^{100}\cdot\mathrm{n}$. Formally, this algorithm is feasible since it's worst case computation time is bounded by a polynomial. However, even for a small n, for example 1, the number of steps required for the algorithm to finish, namely 10^{100}, would be unfeasible.

Let us use a fuzzy approach. The informal description of practical feasibility above uses the notion of something being reasonable. However, this notion is not precise; some lengths and some times are feasible to some extent, but it is unclear as to which extent that is. To describe these notions in a more concrete manner we can:

- assign, to each possible length n the degree $\mathrm{L}(n)$, which ranges between 0 and 1, to which this input length is reasonable: 1 means that we are absolutely sure that n is a reasonable length, 0 means that we are absolutely sure that n is not a reasonable length, and values between 0 and 1 correspond to intermediate degrees of confidence.

- Similarly, we can assign, to each possible time t, the degree $\mathrm{R}(n)$ to which this time is reasonable.

This approach – going back to Lotfi Zadeh – is known as the *fuzzy approach.*

An algorithm with worst-case complexity $\mathrm{T}(n)$ is feasible if for every n for which the corresponding length is reasonable, the time $\mathrm{T}(n)$ is also reasonable, i.e., for all n, we have the implication $\mathrm{L}(n) \implies \mathrm{R}(\mathrm{T}(n))$. In other words, we have the implication corresponding to $n = 1$ and the implication corresponding to $n = 2$, etc.:

$$(\mathrm{L}(1) \implies \mathrm{R}(\mathrm{T}(1)))\ \&\ (\mathrm{L}(2) \implies \mathrm{R}(\mathrm{T}(2)))\ \&\ \ldots\ \&\ (\mathrm{L}(n) \implies \mathrm{R}(\mathrm{T}(n)))\ \&\ \ldots$$

The fuzzy approach allows us to describe the degree to which this statement is true:

- the easiest way to describe the degree to which the statement A & B is true is as min(A, B), and

- the easiest way to describe the degree to which the implication $\mathrm{A} \implies \mathrm{B}$ is true is as max(B, 1 – A).

The last expression comes from the fact that:

- in the usual logic, $\mathrm{A} \implies \mathrm{B}$ is equivalent to $\mathrm{B} \vee \neg\mathrm{A}$,

- the degree of belief in "not A" is naturally described as 1 – A, and

- the degree of belief in "or" of two statements can be estimated as the maximum of the degrees of belief in the two statements.

By using these formulas, we can describe:

- the degree to which statement L(n) $\implies$ R(T(n)) is true by the expression max(R(T(n)), 1 – L(n)), and
- the resulting degree d to which an algorithm with time complexity T(n) is feasible as d = minmax(R(T(n)), 1 – L(n)): n = 1, 2, . . .

How can we compute this degree. The larger the n, the smaller the degree L(n) that n is a reasonable length, so the larger the difference 1 – L(n). On the other hand, the larger the n, the larger T(n) becomes and thus, the smaller the degree R(T(n)) that the time T(n) is reasonable.

Thus, if R(T(n_0)) < 1 – L(n_0) for some n_0, then this inequality is true for all larger values n as well: since with an increase in n, the value R(T(n)) will become even smaller and the value 1 – L(n) will become even larger. For all these values, the maximum is thus equal to max(R(T(n)), 1 – L(n)) = 1 – L(n). This value increases with n, so its minimum is attained for n = n_0.

Similarly, if R(T(n_1)) > 1 – L(n_1) for some n_1, then this inequality is true for all smaller values n as well: since with a decrease in n, the value R(T(n)) will become even larger and the value 1 – L(n) will become even smaller. For all these values, the maximum is thus equal to max(R(T(n)), 1 – L(n)) = R(T(n)). This value decreases with n, so its minimum is attained for n = n_1. Thus, to find the degree d, it is sufficient to consider the first value f for which R(T(f)) < 1 – L(f), and to consider only two values $n = f$ and $n = f$ – 1: d = min(R(T(f – 1)), 1 – L(f)).

2 Practical Feasibility: A Numeric Example

2.1 Determining if an algorithm is practically feasible using fuzzy logic

We can determine if an algorithm with a worst-time complexity of $t(n)$ is practically feasible with a degree of confidence $D(t)$ using the following equation:

$$D(t) = \begin{cases} min(R(t(n_0 - 1)), 1 - R(n_0)) & \text{if } d_0 > 1 \\ 1 - R(1) & \text{if } d_0 = 1 \end{cases} \tag{1}$$

where

- $D(t)$ is the degree of confidence that the input algorithm is practically feasible
- $R(n)$ is the degree of confidence that "n is reasonable"
- n_0 is the first value for which $R(t(n_0)) \leq 1 - R(n_0)$

2.2 Defining A Membership Function for R(n)

Let's come up with a membership function for $R(n)$, the degree of confidence that "n is reasonable".

- We cannot have negative numbers for n, as the input size must be 0 or greater.
- For $n = 0$ (input is empty), although not useful for practical purposes, we will say it's reasonable. Therefore $R(0) = 1$.

- 2^{100} is not reasonable so $R(2^{100}) = 0$. Any amount past this amount of steps would also not be reasonable.
- If the input size can be represented using 16, 32, or 64 bits it has a high probability of being practical as most modern computers can easily process these bit sized numbers. The maximum number than can be represented with X bits is $2^X - 1$. Therefore let's set $R(2^{16}) = 0.9$, $R(2^{32}) = 0.8$, and $R(2^{64}) = 0.7$. We can use linear interpolation to fill the missing values to define the membership function for all values.
- Inputs between $n = 0$ and $n = 2^{16}$ are reasonable, let's set them all to 0.99

Let's define the standard linear interpolation operation (*lerp*) so we can use it to define our membership function.

$$lerp(x_1, y_1, x_2, y_2) = y_1 + (x - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

The membership function therefore is

$$R(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0.99 & \text{if } 0 < x < 2^{16} \\ 0.9 & \text{if } x = 2^{16} \\ lerp(2^{16}, 0.9, 2^{32}, 0.8), & \text{if } 2^{16} < x < 2^{32} \\ 0.8 & \text{if } x = 2^{32} \\ lerp(2^{32}, 0.8, 2^{64}, 0.7), & \text{if } 2^{32} < x < 2^{64} \\ 0.7 & \text{if } x = 2^{64} \\ lerp(2^{64}, 0.7, 2^{100}, 0), & \text{if } 2^{64} < x < 2^{100} \\ 0 & \text{if } x \geq 2^{100} \end{cases} \tag{3}$$

2.3 Results

Using the method and equations described above, we computed the degree of confidence $D(t)$ for different worst-time complexities $t(n)$. The results are shown in the table below.

$t(n)$	n_0	$D(t)$
1	1249541305939518220676374724608	0.99
2^{100}	1	0.01
n	3621858857926702197427908116 48	0.5
n^2	951554188170016	0.2
$1/n$	1249541305939518220676374724608	0.99
$1/n^2$	1249541305939518220676374724608	0.99
$(10^{12})n^2$	1021512950	0.12

Acknowledgments