Towards a More Realistic Definition of Feasibility

Douglas Schirmer[1,2] and Vladik Kreinovich[1]

I. Formulation of the Problem

It is well known that not all algorithms are feasible (see, e.g., [?, ?, ?]): whether an algorithm is feasible or not depends on how many computational steps this algorithm requires. If for input of length n, the algorithm requires 2^n steps, then for an input of a reasonable length $n \approx 300$, we would need more computational steps than there can be during the lifetime of our Universe. Such algorithms (called *exponential-time*) are usually considered not feasible. If, however, the number of elementary steps grows only as a polynomial of n (i.e., an algorithm is *polynomial-time*), then the algorithm is usually considered feasible. In mathematical terms, this leads to a following definition:

Definition 1.

- *For every algorithm U, by its time complexity $t_U(n)$ we mean the largest number of computational steps $t_U(x)$ that this algorithm requires on inputs of length $\le n$.*
- *An algorithm U is called feasible if its time complexity is bounded by some polynomial $P(n)$, i.e., if $t_U(n) \le P(n)$ for all n.*

Comment. In other words, an algorithm U is feasible iff there exists a polynomial $P(n)$ such that for every input x, the number $t_U(x)$ of computational steps of this algorithm does not exceed $P(|x|)$, where by $|x|$, we denoted the length of the input x (e.g., the number of bits that form an input).

This definition is not perfect: It is well known that there are cases when an algorithm is polynomial-time but not feasible (e.g., if the running time is $10^{300} \cdot n$), and vice versa [?, ?, ?], but the above definition is the best that we can get in formalizing "feasible".

There is, however, one more problem with the above definition that is usually not mentioned in feasibility discussions: what we actually want to impose on the algorithm is that its *running time* is small. Instead, we restrict the number of computational steps. This substitution implicitly presupposes that all steps take approximately the same time. This is definitely true at any given moment of time. However, when we distinguish between feasible and non-feasible algorithms, we consider the *asymptotic* case, i.e., the case when the size n and (consequently) the algorithm's running time both tend to infinity. In other words, we consider computations that are performed further and further in the future. As of now, the further into the future we go, the faster are the computations (the time required for one computational step has been so far decreasing exponentially). So, the longer the input (i.e., the larger n), the faster the corresponding computations will be performed.

The goal of the present paper is to take this technological progress into consideration, i.e., to come up with an *optimistic* definition of feasibility.

II. Motivations, Definitions and the Main Result

A. Motivations

When is an algorithm feasible? Informally, feasible means that the time that is required from an algorithm does not exceed the time that the user can wait. This "waiting time" depends on the input. There is some time t_0 that a user is willing to wait if there is not input at all (e.g., to compute the value of a mathematical constant). If the input is large, then simply to enter this input, we need lots of time, and therefore, a user has to wait longer. A reasonable estimate is: If for an input that takes t seconds to enter, the user can wait for additional $t_{\rm add}$ seconds (totally, $t_{\rm add} + t_0$), then for an input that takes ten times longer ($10t$ seconds) to enter, we can wait ten times longer (up to $t_0 + 10 t_{\rm add}$ seconds). In other words, the waiting time $t_{\rm wait}$ cannot exceed t_0 plus a constant C (in the above example, $t_{\rm add}/t_0$) multiplied by the input time $t_{\rm in}$: $t_{\rm wait} \le t_0 + C \cdot t_{\rm in}$.

The speed with which we input and the speed with which we perform computations increases exponentially with time. In other words, the number

[1]Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, email {schirmer,vladik}@cs.utep.edu.
[2]Research Analysis & Maintenance, Inc., 1790 Lee Trevino Dr., Suite 600, El Paso, TX 79936-4525, email raminc@primenet.com.

of operations per second increases as

$$v(t) = v \cdot e^{\alpha t} \qquad (1)$$

for some $\alpha > 0$. Therefore, if initially we could perform $v(0) = v$ operations per second, then after the first second, we can perform $v(1) = ve^{\alpha}$ operations; after two seconds, $v(2) = v \cdot e^{2\alpha}$ operations, etc, and during time t, we can perform

$$N(t) = \int_0^t v(s)\,ds \qquad (2)$$

operations. We arrive at the following definition:

B. Definition

Definition 2. *Let real numbers $v > 0$, and $\alpha > 0$ be fixed.*

- *Let an integer n be given (called length of the input). By an input time, we mean the number $t_{\rm in}$ for which $N(t_{\rm in}) = n$ (where $N(t)$ is defined by the formulas (1) and (2)).*
- *By a computation time of an algorithm U, we mean the number $t_{\rm comp}$ for which $N(t_{\rm comp}) = t_U(n)$.*
- *For given real numbers $C > 0$ and $t_0 > 0$, we say that an algorithm is $(C, t_0)-$feasible if for every n, $t_{\rm comp}(n) \le t_0 + C \cdot t_{\rm in}(n)$.*
- *We say that an algorithm is optimistically feasible if it is $(C, t_0)-$feasible for some C and t_0.*

C. Main Result

Theorem.

- *An algorithm is $(C, t_0)-$feasible for some t_0 if and only if there exists a constant C_U for which $t_U(n) \le C_U \cdot n^C$ for all n.*
- *An algorithm U is optimistically feasible if and only if it is feasible in the sense of Definition 1 (i.e., if and only if it is time-polynomial).*

Comment. This result explains why time-polynomial is usually identified with feasible, and how dependent this identification is on the actual progress. If the progress stops, then "optimistically feasible" will mean linear time. This result also explains the role of the constants C and t_0: if we can wait for a time at least three times larger than the input time, then we can use cubic algorithms; if we can wait 10 times longer, then all algorithms with $t_U(n) \le c \cdot n^{10}$ are feasible, etc.

III. Idea of the Proof

By definition,

$$N(t) = \int_0^t v \cdot e^{\alpha s}\,ds = v \cdot \frac{e^{\alpha t} - 1}{\alpha}.$$

Hence,

$$e^{\alpha t} = \frac{\alpha N(t)}{v} + 1,$$

and

$$t = \frac{1}{\alpha} \cdot \ln\left[\frac{\alpha N(t)}{v} + 1\right].$$

Therefore, the inequality $t_{\rm comp} \le C \cdot t_{\rm in}$ is equivalent to

$$\frac{1}{\alpha} \cdot \ln\left[\frac{\alpha t_U(n)}{v} + 1\right] \le t_0 + C \cdot \frac{1}{\alpha} \cdot \ln\left[\frac{\alpha n}{v} + 1\right].$$

Multiplying both sides by α and turning to exponents, we transform this inequality into the following one:

$$\frac{\alpha t_U(n)}{v} + 1 \le K \cdot \left(\frac{\alpha n}{n} + 1\right)^C,$$

where $K = e^{t_0}$. Therefore, we have

$$t_U(n) \le \frac{v}{\alpha} \cdot \left[K \cdot \left(\frac{\alpha n}{v} + 1\right)^C - 1\right]. \qquad (3)$$

So, if U is $C-$feasible, then $t_U(n)$ is bounded by a polynomial const$\cdot n^C$. Vice versa, if $t_U(n) \le C_U \cdot n^C$, we can show that it satisfies the inequality (3) for an appropriate C_0 ($C_0 \approx \ln(C_U)$).

Acknowledgments. The authors were partially supported by NSF Grants No. CDA-9015006 and No. CDA 9522903, and by NASA Grant No. NAG 9-757.

References

[1] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. F. Freeman, San Francisco, 1979.

[2] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Inc., New Jersey, 1981.

[3] J. C. Martin, *Introduction to languages and the theory of computation*, McGraw-Hill, N.Y., 1991.