VII Международная научная конференция
«Математическое и компьютерное моделирование»
Омск, ОмГУ, 22 ноября 2019 г.

УДК

V. Krymsky[1] and V. Kreinovich[2]
[1]Ufa State Petroleum Technological University
Ufa, Russian Federation
vikrymsky@gmail.com
[2]University of Texas at El Paso,
El Paso, Texas, USA
vladik@utep.edu

HOW HARD IS A GIVEN GENERAL PROBLEM? A PRACTICTIONER'S VIEW AND ITS RELATION TO NP-HARDNESS AND TO AVERAGE COMPUTATION TIME

How hard is a given general problem? Some general problems are computationally feasible, in the sense that there are efficient algorithms that solve all instances of this general problem in reasonable time. Example of such problems include solving systems of linear equations, optimizing a linear objective function under linear equality and inequality constraints (*linear programming*), and many others.

Other problems are harder. How do we gauge this hardness? A typical answer to this question – produced for many such hard problems – is to prove that the problem is NP-hard (technical details can be found, e.g., in [1]). Computer scientists often stop at this step, but is this what practitioners want form them? Our answer is "no". Let us explain what we mean.

What NP-hardness means: a brief reminder. In a nutshell, NP-hardness mean that unless P = NP (which most computer scientists believe to be false), no algorithm is possible to solve all the instances of this general problem in feasible (polynomial) time.

Let us describe this statement in precise terms. Let t(x) denote the running time of an algorithm on input x. The existence of a polynomial-time algorithm would mean that there exists a polynomial P(n) for which, for every input x, we have t(x) <= P(len(x)), where len(x) denotes the length of the input. Thus, the fact that no such polynomial is possible means that for every algorithm that solves the given general problem, for every polynomial P(n), there exists an input x of length len(x) = n for which t(x) > P(n).

In other words, this means that *some* instances of the given general problem are hard.

Is this what practitioners want? Practitioners understand that sometimes things go wrong. Even with a perfect algorithm, a computer may malfunction. This is especially true for algorithms used in real-life systems: systems fail; ideally they fail very rarely, but they do fail.

From this viewpoint, the fact that for *some* instances the problem is hard is no big deal. If these instance are rare – e.g., as rare as computer faults – this is probably OK, a practitioner would not call such a problem computationally hard if for all other cases, the problem can be efficiently solved.

Is average time complexity an answer? Crudely speaking, NP-hardness means that the worst-case complexity is high, i.e., that the worst-case time $\max\{t(x): \text{len}(x) = n\}$ corresponding to all inputs of size n grows, with n, faster than any polynomial.

A natural alternative – well-explored by computer scientists – is to consider *average-time* complexity, i.e., the expected value a(n) of t(x) over some probability distribution (describing how frequently such inputs occur in practice). For example, a well-known simplex method for solving linear programming problems requires, in the worst case, exponential time, but its average computation time is quite reasonable – and indeed, the simplex method works well in most practical problems.

But is this always a good answer? Not necessarily. If for some instances, the computation time t(x) is very long, this will skew the mean value a(n) even if the probability of such cases is very small. And from the practical viewpoint, this makes no sense: if for some instance x, computation goes for too long a time, we do not need to follow it for millions of years: we just stop computations. So what *is* a good measure of hardness?

What is good measure of hardness? A better measure of hardness is the probability that a problem is hard. If for every polynomial P(n), the probability that $t(x) > P(\text{len}(x))$ for inputs of length n exceeds a certain acceptable threshold value p, this means that the problem is *hard* for a significant portion of inputs.

If this probability always tends to 1 as n increases, this means that this problem is *very hard*.

Tasks and challenges. We have definitions. Now it is necessary to analyze which general problems are hard in the sense of these definitions.

This is much more difficult than proving NP-hardness: for that, it is enough to reduce a known NP-hard problem to a given one and thus show that *some* instances of the given problem are hard. However, this will not work for proving the above-defined practical hardness: even when some instance are hard, these instances may be rare and thus, the problem may turn out to be practically feasible.

References

1. *Papadimitriou C.*, Computational Complexity.-- Reading, Massachusetts: Addison-Wesley, 1994.