Results on High Complexity and Diagonal Algorithms are Irrelevant to Computational Practice of Nowadays

Anatol Slissenko
University Paris 12
`slissenko@univ-paris12.fr`

Practical computations should be efficient. We try to apply an algorithm that seems good to solve a given problem, but often the algorithm proves to be practically inefficient. We try to find another one or to improve the initial one analyzing the experimental data, appealing to our intuition and looking at the complexity of the algorithm or that of the problem. The question of the computational complexity of a problem is about the complexity of the 'fastest' algorithms that solve the problem. But what complexity we are speaking about ? That could be the worst case complexity or the average case complexity. For the average case complexity we need some ideas about the distribution of inputs that is usually not so simple. Thus, more often we consider the worst case complexity. For many problems we know their absolute or relative 'high' computational complexity. More precisely, for some problems we know that they are undecidable, or that they are decidable but have exponential or higher lower bounds. For other problems we know that they are relatively difficult, say, NP-hard or NP-complete.

The question is what a 'high lower bound', as mentioned above, means. Let us look at the existing proofs of these high lower bounds. All of them, as far as I am aware, say that there is a subset of inputs that model the calculations of a diagonal Turing machine (or other diagonal algorithm) over all inputs or, that is equivalent, that model the calculations of some set of diagonal Turing machines for a fixed input, for example, for an empty string. And any of these results about high lower bounds says nothing about the complexity on other inputs. But do we really have input instances of this nature in practical computations of nowadays? The answer is definitely negative, sure, modulo the scope of my knowledge (see remarks below on what is diagonal algorithm). Moreover, as far as I know no *practical* algorithm uses diagonal constructions[1].

As an example consider how one typically proves that it is undecidable whether a given program contains a malicious code. Take any program that do some prescribed work. Append into it a piece of code that is executed 'in parallel' with the main activity and that checks self-applicability of some other code X, i. e. this parallel activity applies a universal machine that takes X as program and applies it to X. If this process halts then our appended program behaves as a malware, otherwise it does not affect the main program. As a mass problem over X, it is undecidable whether the malware will be launched or not. But it is clear that if we know what is our initial program about, and what is its structure we can find even by a static analysis a presence of a suspicious part in it. And this is sufficient for the detection of a possible insecurity.

Look at instances of hard problems that arise in practice. Propositional formulas often

[1]One can imagine some algorithms based on diagonal constructions. The most fascinating example that I know, is L. levin's 'optimal algorithm' for the SAT (propositional logic satisfiability) problem – one of the basic NP-complete problems. The algorithm is optimal modulo a multiplicative constant; a brilliant 'cheating' is in the size of this constant.

describe a functioning of some hardware or software. This hardware or software implements algorithms that we understand well like those that are described in manuals or books of reference like in D. Knuth's "The Art of Computer Programming". All these algorithms are far from being diagonal, and thus the propositional formulas modeling them or their abstractions neither. Similar impression one gets from practical instances of optimization problems that often have some structure that permits to decompose them.

Hence, the negative results on high computational complexity, whatever be their great theoretical importance, says nothing about the complexity of practical problems. So it is no astonishing that many algorithms applicable to practical instances of 'hard' computational problems are very efficient in practice.
<u>Thesis</u> ([Sli03]):***The existing negative results about complexity (undecidability, high lower bounds, hardness,...) are not relevant to practical computational problems.***

One of the major 'eternal' problems of computer science is to describe practical computational problems. It does not seem to be a pertinent question for all kind of computational problems. For example, for the multiplication of numbers the question is not pertinent nor feasible, though during the foreseeable future with all existing and future computers only a tiny set of binary numbers of the length 128 will be multiplied if to stick to the known realistic principles of computing. On the other hand, this question looks very pertinent for hard problems. Why we succeed to solve practical computational problems? Because they are sufficiently 'smooth' and not diagonal.

What is "diagonal algorithm" ?

The straightforward observation observation says that a diagonal algorithm is an algorithm that applies its program to itself, and according to some "termination condition" outputs a result that is different from the 'normal' result of this application if the algorithm arrives at it within the mentioned "termination condition". What the question is whether this definition is sufficiently general.

References

[Sli03] A. Slissenko. Complexity problems in the analysis of information systems security. In V. Gorodetsky, L. Popyack, and V. Skormin, editors, *Proc. of the 2nd Intern. Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS-2003), St.Petersburg, Russia, September 21–23, 2003, Lect. Notes in Comput. Sci, vol. 2776*, pages 48–57. Springer-Verlag, 2003. (Invited talk.).