Three Eternal Problems of Theoretical Computer Science

Anatol Slissenko
LACL, University Paris 12

Do we really understand:

1. What is a realistic, practical computer science Problem?
2. What is a realistic, comprehensible and implementable Algorithm?
3. How to evaluate constructively practical Feasibility of a given problem within given computational resources?

Not at all!

What is a realistic, practical computer science Problem?

Example 1. Multiplication of binary numbers.
Multiplication of two 128-bit numbers (2^{256} pairs).
What percentage of these numbers may appear in our computations during one million years?
Number of seconds in 1 million years : $< 2^{45} sec$.
Number of multipliers: each of 10 billions of inhabitants has 1 billion of multipliers making $10^{15} < 2^{55}$ multiplications per second.
We get $2^{(45+70+55)} = 2^{170}$ multiplications.
Thus the amount of numbers that will be multiplied during 1 million years is less than 2^{-86} of all numbers.
Can we describe this meager set to accelerate the multiplication?
Non a minor idea! Hence, the problem is well formulated.

Example 2. Verification in applied predicate logic.

We can describe runs of many practical algorithms and their properties, for example, in the following logic $\mathfrak{P}$:

Presburger arithmetics + abstract functions ($\mathbb{N}$ can be used also to express time).

This logic $\mathfrak{P}$ is undecidable (and incomplete).

Claim: Negative algorithmic results (undecidability, high lower bounds, NP-hardness etc.) are irrelevant to practical computing.

Why? – Because they are based on instances describing application of diagonal algorithms to some inputs.

Such instances never appear in practice.

What instances appear in practice?

They can be described as 'small' theories (a theory is an arbitrary set of formulas) but not closed with respect to traditional syntactic constructions.

A toy example: $\exists\,(a \leq x \cdot N + c \cdot N < b)$, where a, b, c, N are parameters.

This is a formula of arithmetics that is undecidable. But one can easily eliminate quantifiers from it: $\frac{a}{N} < \frac{b}{N}$.

A quantifier free theory (presumably decidable) sufficient for the verification of clock synchronization of N (parameter!) clocks is described in Fundamenta Informaticae, 62(1), 2004.
Other 'small' theories are under construction.

What could be general considerations to define practical classes of instances of this or that problem?

Semantical considerations seem to be the most productive as a starting point.
For example, generalizations of finite model property due to Beauquier–Slissenko.

As for syntactical means, new, special grammars may work.
More developed than, for example, Slisenko's graph grammars for traveling salesman.

This question of description of practical instances is closely related to the next issue, namely, what is algorithm.

What is a realistic, comprehensible and implementable Algorithm?

Only a tiny amount of Boolean functions or of algorithms is comprehensible and implementable.

Mathematics does not study arbitrary functions, nor arbitrary continuous functions, nor even arbitrary smooth functions. It studies particular, always rather smooth, manifolds on which acts a group with some nice properties.

But an algorithm is not only a function, it is more a set of runs. So if we try to look at algorithms from the viewpoint of manifolds, then we have a choice:

- either to study the set of runs of one algorithm as a 'manifold',

or

- to study a 'manifold' of sets of runs of a class of algorithms.

What could be our guidelines towards defining such 'manifolds'?

An algorithm without proof of its properties is useless.
A proof of a property Φ of an algorithm $\mathcal{A}$ is a proof of a formula $\Phi_{Runs\,\mathcal{A}} \longrightarrow \Phi_{Prop}$.
In set-theoretic words: $Runs(\mathcal{A}) \subseteq Models(\Phi)$.
Such a proof evaluates the information contained in $Runs(\mathcal{A})$ with respect to Φ.
If $\Phi = true$ then the proof gives no information. So all depends of the precision of the inclusion above.

The complexity of such a proof can be considered as a parameter (e.g., this permits to speak about complexity of solving an individual instant) describing the information complexity of the algorithm.

Unfortunately, I do not know a 'universal' inference system that may play a role similar to the role of universal function in Kolmogorov complexity.

A set of runs as a compact metric space.

There is another approach, more close to traditional geometry approach.

Consider the runs of an algorithm $\mathcal{A}$ as a set of traces (i.e., a set of sequences of *events*).

Let τ_0 and τ_1 be two traces of $\mathcal{A}$.
Look for maximal *'similar'* sub-traces of τ_0 and τ_1.
'Similar' means 'isomorphic' under an isomorphism that preserves the causal order, 'types' of commands.
Throw away the 'similar' parts, and take as distance the number of events in what rests. This gives a (pseudo-)metric $\beta(\tau_0, \tau_1)$.

This metric gives a locally compact space that suffices to define ε-entropy of its compact subspaces and look at its limit behavior.
Or, we can try to make of this space a compact space.

A sequences of traces $(T(n))_n$ is *convergent* if
$T(n) \leq T(n+1)$,

$\lim_{n\to\infty} |T(n)| = \infty$ and

$\forall\, k\ \lim_{n\to\infty} \frac{\beta(T(n),T(n+k)}{|T(n)|+|T(n+k)|} = 0$.
A convergent sequence of traces consists of asymptotically similar traces.

Modulo some details (related to mutual embedding of the sequences of traces) the *limit metric* between two convergent sequences of traces $(T(n))_n$ and $(T'(n))_n$ is:

$$\delta(T, T') = \limsup_{n\to\infty} \frac{\beta(T(n),T'(n))}{|T(n)|+|T('n)|}$$

With this metric (*still unripe*) the set of traces of $\mathcal{A}$ becomes a compact space and we can define its ε-entropy.
And this gives a *measure of information in the set of traces* that does not depend on any proof of properties of $\mathcal{A}$

The value of ε-entropy of an algorithm $\mathcal{A}$ describes a 'diversity' of ways of processing (better to say, of extracting) the information contained in inputs.

A 'small' ε-entropy corresponds to straightforward algorithms, e.g., string-matching by direct trials, SAT by simple exhaustive search etc.
One can also say that an algorithm with 'small' ε-entropy is in some way smooth.

The observation above, up to now quite informal, gives an idea to attack the feasibility.

How to evaluate constructively practical Feasibility of a given problem within given computational resources?

According to the viewpoint briefly and vaguely outlined above, one can say that *we have not a single result on lower bounds of computational complexity relevant to practical computations.*

One of the reasons lies in the fact that we try to prove lower bounds for the general notion of algorithm. It seems that we do not understand well the power and intricacy of general algorithm.

All practical algorithms have rather small ε-entropy.
Hence, one may try to prove lower bounds for algorithms with a bounded ε-entropy, and then try to extend them to any algorithm.

Other, more clever notions of smoothness may prove to be more productive.

The End

Merci pour ce colloque!