Measuring Computational Complexity: the Qualitative and Quantitative Intertwining of Algorithm Comparison

Marco Benini[*]
Dept. of Pure Mathematics
School of Mathematics
University of Leeds, UK
M.Benini@leeds.ac.uk

Federico Gobbo
DISIM – Dep. of Inf. Eng.,
Comp. Sc., and Mathematics
University of L'Aquila, Italy
federico.gobbo@univaq.it

Manuscript submitted to the Fifth Workshop on the Philosophy of Information, 2013

Abstract

This paper wants to discuss an aspect of information in Computer Science (CS) that is quantitative and qualitative at the same time: measuring. It is a fact that measuring is the basic act of modern science first and engineering later. Also in the common use of the word, measuring is often described as 'describing a phenomenon by a number'. Nevertheless, in Computer Science, this meaning is not always respected, in particular in the field of computational complexity—an essential part of CS.

The theory of computational complexity is the branch of theoretical CS which studies the efficiency of algorithms in terms of time, i.e., the number of steps to perform a complete computation, and space, i.e., the quantity of memory cells which are needed to obtain a result. Of course, an algorithm A is more efficient than an algorithm B when it uses less the resource we want to minimise—either time or space.

Here, a relevant problem arises: the time (and the space) needed to perform a computation depends on the input. Researchers, see, e.g., (Papadimitriou, 1994), in CS have abstracted over the input by considering only its size: therefore, choosing between two algorithms A and B means to consider the worst performance of an algorithm A for an input of size n, the worst performance of an algorithm B for an input of the same size, and to extend this comparison to every n. Thus, A is more efficient than B if, **for every** n, the time A spends to compute the result on the worst input of size n is less than the time B uses to perform the same task on its worst input of size n. The abstraction means that the complexity of an algorithm A becomes a function, assigning to each number n the time needed by A to compute the result on its worst input of size n.

When we ask to compare two algorithms, we really want to compare their complexities; the long-term behaviour is then captured by saying that one function *definitely dominates* the other one, using the jargon of mathematical analysis. In this way, experts really measure efficiency of algorithms using functions instead of numbers—in

[*]Dr Benini was supported by a Marie Curie Intra European Fellowship, grant n. PIEF-GA-2010-271926, *Predicative Theories and Grothendieck Toposes*, within the 7th European Community Framework Programme.

other words, the nature of the information produced hereby is not completely quantitative, if we follow the definition stated at the beginning of this paper.

As a result, in most real-world cases algorithms result to be incomparable in practice, as each one may be more efficient than the other one for some values of n. So, researchers turned around this problem by observing that what really matters is the long-term behaviour of an algorithm. The fundamental question is: when n is "big enough", which algorithm under consideration performs better?

We consider the act of measuring the complexity of a pair of algorithms as an act of producing information, where 'information' is intended in the sense used within the Philosophy of Information (PI).[1] The act is performed by an informational organism (inforg), where the biological agent is the algorithm expert, the engineered artifact being the measure of complexity of the algorithms under scrutiny, which are their primary Level of Abstraction of interest. It is worth noticing, that this algorithmic inforg eventually generates an indefinite number of computational inforgs, i.e., inforgs whose artificial counterpart is a kind of computing machine based on the algorithm chosen, typically based on Von Neumann's model (Gobbo and Benini, 2013). However, the software produced over the algorithms is not part of their 1^{st} order ontological commitment (i.e., their knowable observables, in terms of relational structures) but possibly to a 2^{nd} order ontological commitment (Floridi, 2011, 351–352).

Furthermore, the pieces of information collectively produced by the algorithmic inforgs gives a new Level of Explanation (LoE), a sort of second-order *qualitative* change in the classification of algorithms themselves. According to their ontological commitment, algorithmic inforgs observe, compare, and finally classify algorithms, i.e., their complexities, an activity which can be carried on only by the human part of the inforg (Gobbo and Benini, forthcoming). It emerged that sometimes the complexity associated to some algorithms exhibit a behaviour which is similar to the one of a polynomial; in some other cases, the complexity grows much faster, as the exponential function or even more. This fact introduces the new LoE, which classify algorithms in two parts: the ones which are 'feasible' (polynomial) and the ones which are 'practically uncomputable' (exponential). Again, this LoE is both qualitative and quantitative. This distinction is codified in the so-called *extended Church-Turing thesis* which says that the only practically computable problems are the ones admitting a polynomial algorithm able to compute them.

The search for the borderline between feasible and unfeasible algorithms is still unclear. In fact, each polynomial algorithms operating on a non-deterministic Turing machine can be easily simulated on a deterministic machine by trying all the possible non-deterministic paths: the complexity of the simulated algorithm is exponential. However, a subclass of those algorithms, the 'decision procedures', is particularly relevant. This class is called **NP** in literature and the problem if **NP** is distinct from **P**, the class of polynomial algorithms on a deterministic machine, is the most important and well-known open problem in theoretical CS, thus an eminently qualitative problem whose solving attempts have generated an immense amount of knowledge and ideas in the last 40 years, see, e.g., (Savage, 1998).

In the sequel of this paper, we will explore some consequences of the conducts of algorithmic inforgs, which go beyond the scope of CS. In fact, as effectively put by Barry Cooper: 'algorithms, as a way of traversing our four dimensions, have been with us for literally thousands of years. They provide recipes for the control and understanding of every aspect of everyday life (Cooper, 2012, 776).'

[1] In this paper, we take as granted the fundamental notions of PI, as explained in Floridi (2011).

References

Cooper, S. B. (2012), ‘Incomputability after Alan Turing’, *Notices of the AMS* **59**(6), 776–784.

Floridi, L. (2011), *The Philosophy of Information*, Oxford University Press, Oxford.

Gobbo, F. and Benini, M. (2013), ‘The minimal levels of abstraction in the history of modern computing’, *Philosophy & Technology* pp. 1–17.
URL: *http://dx.doi.org/10.1007/s13347-012-0097-0*

Gobbo, F. and Benini, M. (forthcoming), ‘Why zombies can’t write significant source code: The knowledge game and the art of computer programming’, *Journal of Experimental & Theoretical Artificial Intelligence* .

Papadimitriou, C. H. (1994), *Computational Complexity*, Addison-Wesley.

Savage, J. E. (1998), *Models of Computation: Exploring the Power of Computing*, Addison-Wesley.