Measuring Computational Complexity

the qualitative and quantitative intertwining of algorithm comparison

*Marco Benini

marco.benini@uninsubria.it

Department of Pure Mathematics

University of Leeds

Federico Gobbo

federico.gobbo@univaq.it

Dep. of Eng., Comp. Sc. and Maths.

Università degli Studi dell'Aquila

Introduction

Given P and Q, two computer programs running on the same machine, written in the same language, what does it mean

$$P \text{ is faster than } Q?$$

The *theory of computational complexity* wants to answer this kind of questions, i.e., when a program uses better some resource of a computing device, or when a problem is solvable by an efficient program.

Decision problems

As customary, assume that our programs solve *decision problems*:

- $\langle I, A\rangle$ with I a set of possible inputs, and $A \subseteq I$;
- a program, given an input $i \in I$, terminates with a result which is **YES** when $i \in A$, **NO** otherwise;

Furthermore, we assume that

- programs do terminate for any input $i \in I$;
- programs operate on some *reasonable* machine.

Time as steps

Instead of measuring time by seconds, minutes, hours, we measure time as the number of elementary instructions a program performs in its computation.

- This is an abstraction over the *technology* of the machine, as it reduces time to what is observable in the language the programs are written in;
- From an engineering point of view, this abstraction is reasonable, as elementary instructions execute in similar times;

This is a new *level of abstraction* (LoA, [Floridi (2008)]):

- internally coherent;
- the level of observation is clear;
- designer + machine: time as steps;
- programmer + machine: time as seconds.

Complexity

The *complexity* of a program is a function $C\colon I \to \mathbb{N}$, mapping each input into the number of steps needed to obtain an answer.

We compare two programs by comparing their complexities.
We classify a problem by means of the *best* program which solves it.

- Good idea, bad implementation!
- Complexities are not uniform;
- Paradoxes—if interested in one result you have to calculate it **before** choosing the calculator!

Abstract over the input

- Worst case analysis: fixed the *size* of the input, choose the worst input of that size;
- Complexity is a function $C: \mathbb{N} \to \mathbb{N}$, mapping each size to the longest number of steps needed to obtain a result for an input of that size;

- Globally uniform
- The choice of the worst case can be done statically, estimating an upper bound of the value $C(n)$, for each $n \in \mathbb{N}$.

A new level of abstraction

- Globally uniform: if we are allowed to perform at most k steps, we cannot scan an input whose size is greater than k, that is, *time bounds size*;
- Complexities can be derived by observables: programs are observable, and static analysis on them provide us with (bounds for) complexities.
- a new inforg: analyst + (program +) abstract machine.

Functions, i.e., structured collections of values, instead of measures: we are out of the quantitative world…

Problems

. . . but we are not yet in a qualitative world!

In fact,

- complexities are *too* informative to use for comparisons;
- complexities do not form an order, thus classification of programs/problems is impossible;
- we do not use *real* complexities but an estimation of them, typically an upper bound.

Complexity as a class

Since estimations are used, why not to promote them?

The *complexity* of a program is a function which bounds the *real* complexity

- if C is the complexity of P as previously defined, and F is a function $\mathbb{N} \to \mathbb{N}$ in some predefined class, F is the *complexity class* of P when there are $a, b, k \in \mathbb{N}$ such that $aF(n) \leq C(n) \leq bF(n)$ for every $n \geq k$.

Thus F provides the *growth rate* of the complexity of P for increasing values of the size of the input.

Complexity classes

A good choice of *reasonable* classes is available:

- polynomials: $\sum_{i=0}^{n} a_i x^i$;
- exponentials: $b^{\sum_{i=0}^{n} a_i x^i}$;
- logarithms: $\sum_{i=0}^{n} (a_i x^i + b_i \log^i x)$.

These choices have nice mathematical properties both facilitating the static analysis of programs, and having enough expressive power to estimate every *natural* program.

- these classes are comparable: $n \log x \leq x^n \leq n^x$ for sufficiently large n and x;
- they form a classification.

Extended Church-Turing thesis

Most problems falls into two super-classes:

- **P** is the class of problems solvable by a program operating in polynomial time;
- **EXP** is the class of problems solvable by a program operating in exponential time.

Evidently, **P** is contained in **EXP**. And it is possible to prove that the two super-classes are distinct.

The *extended Church-Turing thesis* says that programs operating in polynomial time are *effective*, while programs operating in exponential time are *practically uncomputable*.

Big problems!

If we allow the machine to do a limited amount of guessing, some problems in **EXP** become polynomial.

Without giving the formal details, on these machines, called non-deterministic, the polynomial problems form a class called **NP**. It is immediate to see that **P** is contained in **NP**, and that **NP** is contained in **EXP**.

The most important open problem in theoretical computer science is whether **P** is distinct from **NP**. All the world bets that this is the case, but. . .

Quality versus quantity

The abstraction “complexity as a function” to “complexity as a class” does **not** form a level of abstraction (LoA):

- no new inforg is needed (and this is a good hint);
- its associated level of observation is identical.

In fact, it is better understood as a different *level of explanation* (LoE), using Luciano Floridi’s terms [Floridi (2008)].

As a LoE, it is completely qualitative, allowing for a classification of decision problems.

In terms of Information Quality (IQ) analysis, complexity as a class is P-purpose, as algorithms serve to *produce* programs, regardless of the actual use of them, i.e., their *consumption* (C-purpose) [Floridi (2013)].

Conclusions

It is clear how the development has been achieved.

There is a poor understanding of the exact meaning of the intermediate level of abstraction, where complexity is measured by a function.

We hope this real case study may lead to some insight on the threshold between quantitative versus qualitative measuring, addressing the P-purpose within the LoE of algorithm comparison.

References

Floridi, L. (2013), “Information Quality,” in *Philos. Technol.*, 26, 1–6.

Floridi, L. (2008), “The method of levels of abstraction,” in *Minds and Machines*, 18, 3, 303–329.

Thanks for your attention!

Questions?

For proposals, ideas & comments:

Download & share these slides here:

`http://slidesha.re/14rFpZD`

marco.benini@uninsubria.it & federico.gobbo@univaq.it