INFR11102: Computational Complexity 17/09/2019

Lecture 1: Introduction

Lecturer: Heng Guo

1 Theory of computing

Structural theory and computational methods have always been two major themes of mathematics. For example, Euclid's *Elements* is mostly structural, building up a theory deductively from axioms and postulates, but it also contains algorithmic gems like the Euclidean Algorithm (for the gcd of two integers). Quite often to prove the correctness and efficiency of an algorithm requires purely structural results.

Arguably, most significant discoveries in modern mathematics are structural. However, a big part of their motivation is computational. Let's look at a few examples: calculus was invented for computing orbits of planets; Galois theory and finite group theory were found via investigating how to solve equations; the Prime Number Theorem was first conjectured by Gauss after much computational experiments. This list goes on and on.

On the other hand, we didn't have a formal model of computation until 1930s. Since its genesis, there has been an amazing amount of development on the theory of computing. The mathematical side of computing has been split into "Theory A" and "Theory B" (these names come from a handbook of theoretical computer science published in 90s, which is split into volume A and B). Theory A is about figuring out how much resources do we need to spend in order to perform a certain computational task. There are many resources of interest. The most common one is the amount of time the computation would require, but we may also be interested in space (memory), randomness, etc. We will be talking mostly about Theory A, and Theory B is more concerned with formal methods, logic, programming languages etc.

At a very high level, there are two main parts of Theory A: algorithms and computational complexity. On the algorithmic side, we want to develop a procedure to solve the problem as efficiently as possible, thus providing upper bounds on the resource spent. On the flip side, computational complexity studies how much resource do we *have to* use, namely proving lower bounds. These two parts are highly connected. As the name suggests, our class will mainly focus on computational complexity.

2 Model of computation

Computer science is all about solving problems. By a computational problem, we mean a function that maps its inputs to its outputs. Let Σ be a finite alphabet, such as $\{0,1\}$ or $\{a,b,c,d\}$. Let Σ^* be the set of strings consisting of symbols from Σ with arbitrary length. Then the computational task is usually a function of the form: $\Sigma^* \to \Sigma$.

Here is an example.

Name: MULTIPLY

Input: Two integers a and b.

Output: Their product ab.

Apparently this is a very simple (really?) task. We learned how to do it in primary school. However, the naive method is terribly inefficient, and it is actually still open what is the most efficient algorithm. Suppose the two integers a and b both have n digits, then the naive algorithm requires $O(n^2)$ operations. However, Schönhage and Strassen [SS71] have shown that using Fast Fourier Transformation (FFT), integer multiplication can be done in time $O(n \log n \log\log n)$. For quite some time, Schönhage and Strassen's algorithm is the state of the art, until Fürer [Für09] improves the running time into $O\left(n \log n 2^{O(\log^* n)}\right)$. Fürer's work inspires a number of follow-up work. The most recent work of Harvey and van der Hoeven [HvdH19] has claimed an almost optimal running time of $O(n \log n)$. This is almost optimal because (also recently) Afshani et al. [AFKL19] showed a lower bound of $\Omega(n \log n)$ for Boolean circuits, assuming a conjecture in network coding.

MULTIPLY is a problem with integer outputs. There is another important class of problems, called *decision problems*, whose output is supposed to be Yes/No. The most common encoding for a decision problem is that the output is $\{0,1\}$. A decision problem is also called a *language*, which is a subset of all possible strings Σ^* so that the answer is "Yes". Namely, for a decision problem $f : \Sigma^* \to \{0,1\}$, define the corresponding language

$$L_f := \{x : f(x) = 1\}.$$

Name: DIOPHANTINE

Input: A multivariate polynomial equation with integer coefficients.

Output: Does the equation have an integer solution?

Examples of DIOPHANTINE include equations like $x^2 - 3xy + 2y^2 = 0$, $x^{10} + y^{10} - z^{10} = 0$, etc.

This is a very famous problem, proposed by David Hilbert in his speech at the second International Congress of Mathematics (ICM) in 1900. (He actually listed 23 important mathematics problems. This one is the 10th.) For quite some time, there was no good answer for this problem. Later in 1928, Hilbert proposed the ENTSCHEIDUNGSPROBLEM (German for "decision problem").

Name: Entscheidungsproblem

Input: A first-order logic statement.

Output: Is the statement valid?

Examples of Entscheidungsproblem are "$\forall x, y, (x = y) \to (x^2 = y^2)$", "$\exists x, \forall y, x = y$", etc.

Again, for almost another decade, there is no solution to this problem. One of the main issues is that there was no good definition for what an algorithm is. In 1936, Alonzo Church [Chu36] introduced λ-calculus as a model of computation, and showed that there is no algorithm for the Entscheidungsproblem. However, λ-calculus was not accepted as a reasonable model and it stirred quite some debate.

At about the same time, Alan Turing independently introduced another model, later named the *Turing Machine* (TM) (published in 1937 [Tur37b]). This model is very natural, and Turing argued in great length that why this is the correct model for a "computer". Note that a "computer" means "a person who computes" from 1640s to 1940s. A short while later, Turing [Tur37a] showed that TM and λ-calculus are actually equivalent. Therefore, it is widely accepted that Entscheidungsproblem is undecidable.

The Church-Turing thesis states that, the notion of an algorithm is precisely captured by TM (or λ-calculus). This is not a theorem, but is widely accepted.

Going back to the Diophantine problem, it turns out that the problem is undecidable as well. This is proved by Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson which spans 21 years, with Yuri Matiyasevich completing the theorem in 1970. See the book [Mat93] for a detailed account.

3 Content of this course

Our course will start with a brief review of the Turing machine, and show that the halting problem is not decidable by diagonalisation. We will then cover the following topics:

1. time/space hierarchy theorems;
2. basic computational complexity classes and their relations;
3. the circuit model, and the polynomial hierarchy;
4. randomised computation and counting complexity.

If time permits, we may also touch some more specialised subjects, such as interactive proof systems and fine-grained complexity.

References

[AFKL19] Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. Lower bounds for multiplication via network coding. *CoRR*, abs/1902.10935, 2019.

[Chu36] Alonzo Church. A note on the Entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

[Für09] Martin Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[HvdH19] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *HAL archive*, hal-02070778, 2019.

[Mat93] Yuri V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[Tur37a] Alan M. Turing. Computability and λ-definability. *J. Symb. Log.*, 2(4):153–163, 1937.

[Tur37b] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(1):230–265, 1937.