M1 ENS Lyon: Computational Complexity **Fall 2014**

Lecture 1 — September 10

Lecturer: Omar Fawzi *Scribe: Omar Fawzi*

1.1 Some admin

See course website http://www.omarfawzi.info/teaching/complexity.

1.2 From computability to complexity

The object of study for this course is the fundamentals of computation. Computation was precisely defined by Turing (and others) in the 1930s. In this course, we study problems that are within the computable. We would like to differentiate between problems in terms of the resources needed to solve them. For example, can we formalize the differences between the tasks of addition and multiplication, of testing whether a graph is 2-colorable and testing whether it is 3-colorable, between solving a Sudoku puzzle and checking that a solution is correct.

1.2.1 Objectives and motivation

Complexity theory is a resource theory for computation. We define relevant resource for computation and study what are the resources needed in order to compute a given computational task. Typical resources of interest include time, space (or memory), communication, randomness, etc... We also study the relation between the different resources: can I improve the running time at the expense of using more memory? We also compare tasks by the resources needed to solve them: given a black box that performs a given task, can I use it to efficiently perform the task I am interested in?

Complexity theory is an essential ingredient in modern cryptography. In fact, most of today's cryptography is based on complexity assumption. One main tool is called a one-way function, which is a function that can be efficiently computed but cannot be inverted in polynomial time. In order to break the protocol, an adversary would need to perform the costly (in terms of computation resources) task of inverting the function.

Taking a broader picture in the sciences, understanding computation is certainly a way of understanding a theory in physics. For example, quantum information has proved quite useful in the understanding of quantum mechanics. We can mention that quite recently researchers studying black hole physics got interested in quantum error correcting codes and quantum complexity classes. [1]

Some of the big questions of complexity theory. As you all know, the **P** versus **NP** question is a fundamental open question in complexity theory, but in general in mathematics. But we are actually very far from solving it. One frontier is showing that circuits with AND, LOR, NOT and MOD$_6$ gates of *constant* depth cannot solve the problem SAT

Another very interesting question is about the power of randomness. To state it in terms of classes, is $\mathbf{P} = \mathbf{BPP}$, or is $\mathbf{L} = \mathbf{RL}$?

Also, for which problems can quantum mechanical computers give speedups? How to compare the class **BQP** of problems solvable in polynomial time on a quantum computer to traditional complexity classes like **P** and **NP**?

1.3 Turing machines and time

The course will follow closely the book [1] and for some parts [2].

We start with the definition of a Turing machine.

Definition 1.3.1 (Turing machine). *A k-tape Turing machine is described by a triple (Γ, Q, δ) with the following properties*

- *Γ is a finite set of symbols (also called alphabet). We assume $\Box \in \Gamma$ is the blank symbol and $\triangleright \in \Gamma$ denotes the beginning of the tape.*
- *Q is a finite set of states, with two distinguished states called $q_{start}, q_{halt} \in Q$*
- *A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$*

How does a computation work? The starts in the state q_{start} and we think of the machine as acting on k infinite tapes (we take them to be infinite only on one side here). The first tape (called input tape and is read-only) is initialized to $\triangleright x_1 \dots x_n \Box\Box \dots$, where $x_1 \dots x_n$ represents the input to the Turing machine. All other tapes are initialized to $\triangleright\Box\Box \dots$. Each tape

[1]http://www.nature.com/news/theoretical-physics-complexity-on-the-horizon-1.15285

has a head that starts at the beginning of the tape. The machine then changes its configuration as follows. Considering its internal state q and the k symbols $\sigma_1, \dots, \sigma_k$ at the positions of the head of each tape, we apply the function $\delta(q, \sigma_1, \dots, \sigma_k) = (q', (\sigma'_2, \dots, \sigma'_k), m)$, where $m \in \{L, S, R\}^k$. q' is the new internal state of the machine and σ'_i is the new symbol that replaces σ_i at the i-th tape in the position given by the head. m dictates the movement of the heads for the next configuration, *L*eft, *S*tay or *R*ight.

Now let us define what it means for a Turing machine M to compute a function f.

Definition 1.3.2. *For a function $f : \Sigma_{in}^* \to \Sigma_{out}^*$, we say that M computes the function f if on all inputs $x \in \Sigma_{in}^*$, the machine M halts on x and $f(x)$ is written on the output tape.*

We say that M computes f in time $T(n)$ if M computes f and for all $x \in \Sigma_{in}^*$, the computation of M on input x halts after at most $T(|x|)$ steps, where $|x|$ refers to the number of symbols of x.

Remark To fully specify a function, we have to specify an encoding. There are usually many possible encodings and these in general can affect the resources used. But mostly for this course, the encoding will not play an important role as long as it is reasonable. For example, the standard encoding for an integer would be a binary encoding, for a matrix it would be the list of its entries and for a graph, it would be its adjacency matrix. Another observation is that the running time of a machine is defined as the worst case over all inputs x of the same size.

Example Consider the function EVEN : $\{0,1\}^* \to \{0,1\}$ with EVEN$(x) = 1$ iff the number whose binary representation is x is an even number. An example of a 2-tape TM that computes EVEN is as follows. The idea is to read the input until we reach the symbol $\Box$ and then come back one cell to figure out the last digit. Let $\Gamma = \{\rhd, \Box, 0, 1\}$, $Q = \{q_{start}, q_{halt}, q_1\}$ and δ is as follows.

$$\forall \sigma \in \{\rhd, 0, 1\} \quad \delta(q_{start}, (\sigma, \rhd)) = (q_{start}, \rhd, (R, R)) \tag{1.1}$$
$$\forall \sigma \in \{\rhd, 0, 1\} \quad \delta(q_{start}, (\sigma, \Box)) = (q_{start}, \Box, (R, S)) \tag{1.2}$$
$$\delta(q_{start}, (\Box, \Box)) = (q_1, \Box, (L, S)) \tag{1.3}$$
$$\forall \sigma \in \{0, 1\} \quad \delta(q_1, (\sigma, \Box)) = (q_{halt}, \sigma, (S, S)) \tag{1.4}$$
$$\delta(q_1, (\rhd, \Box)) = (q_{halt}, 1, (S, S)) \tag{1.5}$$

For the last line, we took the convention that the empty string is in EVEN.

Before moving to the defining our first complexity classes, we highlight some of the important properties of the computational model.

1. The computational model is robust, e.g., the size of the alphabet, the number of tapes but more generally, TMs can simulate all known physically realizable computation devices. The idea that a TM can simulate any such physically realizable device is known as the Church Turing thesis.

2. It is simple to encode a TM $M = (\Gamma, Q, \delta)$ into a bit string. We can thus label TMs by bit strings that correspond to their encoding. We call M_α the TM encoded by $\alpha \in \{0,1\}^*$.

3. There exists a universal TM.

   Theorem 1.3.3. *There exists a TM U such that for every $x, \alpha \in \{0,1\}^*$, $U(x, \alpha) = M_\alpha(x)$. Moreover, if M_α halts on input x within T steps, then $U(x, \alpha)$ halts within $CT \log T$ steps where C is a number independent on $|x|$ and depending only on M_α's alphabet size, number of tapes and number of states.*

1.4 The classes P and NP

A complexity class is simply a set of functions that can be computed within some resource bounds. It is convenient to restrict ourselves to boolean functions (this is especially for nondeterministic classes) $f : \Sigma^* \to \{0,1\}$. Equivalently, one can talk about the language $L = \{x \in \Sigma^* : f(x) = 1\}$.

Definition 1.4.1. *$L \in$ **DTIME**$(T(n))$ iff there exists a constant $c > 0$ and a TM M that runs in time $cT(n)$ that computes L.*

Remarks The reason we allow for this constant c is so that **DTIME**$(T(n))$ does not dependent on the alphabet size of the Turing machine. In fact, it is perhaps not so surprising that by increasing the alphabet size. This is sometimes known as an acceleration theorem, see [2, Theorem 2.E] for more details.

Definition 1.4.2. $\mathbf{P} = \bigcup_{\ell \geq 1} \mathbf{DTIME}(n^\ell)$

Examples

1. Even $\in \mathbf{DTIME}(n) \subset \mathbf{P}$.

2. Given two integers x and y and i, determining whether the i-th bit of the product $x \cdot y$ is 1 can be clearly done in $\mathbf{DTIME}(n^2) \subset \mathbf{P}$. It can actually be done much more efficiently, almost in $O(n \log n)$.[2]

3. Given a graph G and a number k, determining whether G has a matching of size $\geq k$ is in $\mathbf{P}$.

P is generally consider as the class of problems that are efficiently solvable "in nature". Of course, this might be not the relevant notion for many settings in practice. Sometimes, the inputs are so large that a number of steps of $O(n^2)$ is out of the question. In the area of streaming algorithms, researches study a model in which one has only one pass over the input x and a working memory that is, say, $O(\log |x|)$. But let us mention some reasons for which **P** is considered to capture efficiently computable problems.

1. **P** seems quite robust to the model of computation. This is sometimes known as the strong Church Turing thesis, which claims that every physically realizable computation can be simulated by a TM with at most a polynomial overhead. We should note that this idea is a bit controversial especially with the definition of quantum computers.

2. The class **P** is closed under subroutine. We would like to have a definition of efficiency that satisfies the following property: if, in an efficient program, you replace some instructions by efficient subroutines, the overall program should be efficient. The class **P** satisfies this property.

3. Another argument is that for the known problems of interest that have a polynomial time algorithms, they are in $\mathbf{DTIME}(n^\ell)$ for some reasonably small value of ℓ.

We now move to the complexity class **NP** which captures many of the problems that we want to solve. Intuitively, **NP** corresponds to the class of problems for which you can efficiently check whether a solution is valid or not.

Definition 1.4.3 (NP). *$L \in \mathbf{NP}$ iff there exists a polynomial p and a polynomial time machine M such that for every $x \in \{0,1\}^*$,*

$$x \in L \quad \Leftrightarrow \quad \text{there exists } u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1 \ . \tag{1.6}$$

u is called a certificate or witness for x.

[2] http://en.wikipedia.org/wiki/Multiplication_algorithm

Examples

1. Even $\in$ $\mathbf{NP}$, and more generally $\mathbf{P} \subseteq \mathbf{NP}$
2. sat $\in$ $\mathbf{NP}$. This problem will be defined in more detail in the next lecture.

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[2] Sylvain Périfel. *Complexité Algorithmique.* Ellipses, 2014.