PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 1: Introduction

Lecturer: *Sanjeev Arora* Scribe:*Scribename*

The central problem of computational complexity theory is : how efficiently can we solve a specific computational problem on a given computational model? Of course, the model ultimately of interest is the Turing machine (TM), or equivalently, any modern programming language such as C or Java. However, while trying to understand complexity issues arising in the study of the Turing machine, we often gain interesting insight by considering modifications of the basic Turing machine —nondeterministic, alternating and probabilistic TMs, circuits, quantum TMs etc.— as well as totally different computational models— communication games, decision trees, algebraic computation trees etc. Many beautiful results of complexity theory concern such models and their interrelationships.

But let us refresh our memories about complexity as defined using a multitape TM, and explored extensively in undergrad texts (e.g., Sipser's excellent "Introduction to the Theory of Computation.") We will assume that the TM uses the alphabet $\{0,1\}$. A *language* is a set of strings over $\{0,1\}$. The TM is said to *decide* the language if it accepts every string in the language, and rejects every other string. We use asymptotic notation in measuring the resources used by a TM. Let $\mathbf{DTIME}(t(n))$ consist of every language that can be decided by a deterministic multitape TM whose running time is $O(t(n))$ on inputs of size n. Let $\mathbf{NTIME}(t(n))$ be the class of languages that can be decided by a nondeterministic multitape TM (NDTM for short) in time $O(t(n))$.

The classes $\mathbf{P}$ and $\mathbf{NP}$, and the question whether they are the same is basic to the study of complexity.

DEFINITION 1 $\mathbf{P} = \cup_{c \geq 0} \mathbf{DTIME}(n^c)$.
$\mathbf{NP} = \cup_{c \geq 0} \mathbf{NTIME}(n^c)$

We believe that the class $\mathbf{P}$ is invariant to the choice of a computational model, since all "reasonable" computational models we can think of happen to be polynomially equivalent[1]. Namely, t steps on one model can be simulated in $O(t^c)$ steps on the other, where c is a fixed constant depending upon the two models. Thus in a very real sense, the class $\mathbf{P}$ exactly captures the notion of languages with "feasible" decision procedures. Of course, one may argue whether $\mathbf{DTIME}(n^{100})$ represents "feasible" computation in the real world. However, in practice, whenever we show that a problem is in $\mathbf{P}$, we usually can find an n^3 or n^5 time algorithm for it.

[1]Recent results suggest that a computational model based upon quantum mechanics may not be polynomially equivalent to the Turing machine, though we do not yet know if this model is "reasonable" (i.e., can be built). We will discuss the quantum model later in the course.

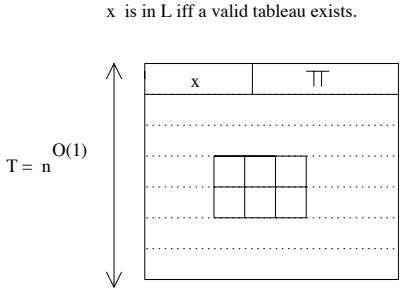

Figure 1: Tableau as used in Cook-Levin reduction

NP contains decision problems in which a "YES" answer has a short certificate whose size is polynomial in the input length, and which can be verified deterministically in polynomial time. Formally, we have the following definition, which is easily seen to be equivalent to Definition 1.

DEFINITION 2 (ALTERNATIVE DEFINITION OF **NP**) *Language L is in* **NP** *if there is a language $L_0 \in \mathbf{P}$ and constants $c, d > 0$ such that*

$$\forall x \in \{0,1\}^* \quad x \in L \iff \exists y \in \{0,1\}^*, |y| \leq |x|^c + d \quad \textit{and} \quad (x,y) \in L_0.$$

EXAMPLE 1 CLIQUE, 3-COLORING are **NP** problems and not known to be in **P**. The language of connected graphs is in **P**.

A *polynomial-time reduction* from language A to language B is a polynomial-time computable function f mapping strings to strings, such that $x \in A$ if and only if $f(x) \in B$. A language is **NP**-*hard* if there is a polynomial-time reduction from every **NP** language to it. An **NP**-hard language is **NP**-*complete* if it is **NP**-hard and in **NP**.

Cook and Levin independently showed that the language 3SAT is **NP**-complete. We briefly recall this classical reduction. Let L be an **NP** language and x be an input. The reduction uses the idea of a tableau, which is a step-by-step transcript whose ith line contains the state of the tape at step i of the computation. Clearly, $x \in L$ iff there exists a tableau that contains a computation of M that contains x in the first line and in which the last line shows that M accepts (Figure 1).

The main observation is that the tableau represents a correct computation iff all 2×3 *windows* look "correct," i.e. thay satisfy some local consistency conditions. Cook-Levin reduction encodes these consistency checks with $3CNF$ clauses so that a valid tableau exists iff the set of all these $3CNF$ clauses is satisfiable.

REMARK 1 The fact that **NP**-hard problems *exist* is trivial: the halting problem is an **NP**-hard problem. Actually, the fact that **NP**-complete languages exist is also trivial. For instance, the following language is **NP**-complete:

$$\{< M, w, 1^n >: \text{NDTM } M \text{ accepts } w \text{ in time } n\}.$$

Cook and Levin's seminal contribution was to describe explicit, combinatorial problems that are **NP**-complete. 3SAT has very simple structure and is easy to reduce to other problems.

1 EXPTIME and NEXPTIME

The following two classes are exponential time analogues of **P** and **NP**.

DEFINITION 3 $\mathbf{EXPTIME} = \cup_{c \geq 0} \mathbf{DTIME}(2^{n^c})$.
$\mathbf{NEXPTIME} = \cup_{c \geq 0} \mathbf{NTIME}(2^{n^c})$.

Is there any point to studying classes involving exponential running times? The following simple result —providing merely a glimpse of the rich web of relations we will be establishing between disparate complexity questions— may be a partial answer.

THEOREM 1
If $\mathbf{EXPTIME} \neq \mathbf{NEXPTIME}$ *then* $\mathbf{P} \neq \mathbf{NP}$.

PROOF: We prove the contrapositive: assuming $\mathbf{P} = \mathbf{NP}$ we show $\mathbf{EXPTIME} = \mathbf{NEXPTIME}$. Suppose $L \in \mathbf{NTIME}(2^{n^c})$. Then the following language

$$L_{\text{pad}} = \left\{ < x, 1^{2^{|x|^c}} > : \ x \in L \right\} \tag{1}$$

is in **NP** (in fact in $\mathbf{NTIME}(n)$). (Aside: this technique of adding a string of symbols to each string in the language is called *padding*.) Hence if $\mathbf{P} = \mathbf{NP}$ then L_{pad} is in **P**. But if L_{pad} is in **P** then L is in **EXPTIME**: to determine whether an input x is in L, we just pad the input and decide whether it is in L_{pad} using the polynomial-time machine for L_{pad}. □

What if P = NP?

If $\mathbf{P} = \mathbf{NP}$ —specifically, if an **NP**-complete problem like 3SAT had say an $O(n^2)$ algorithm— then the world would be mostly a Utopia. Mathematicians could be replaced by efficient theorem-discovering programs (a fact first pointed out by Kurt Gödel in 1955). AI software would be perfect since we could easily do exhaustive searches in a large tree of possibilities. Inventors and engineers would be greatly aided by software packages that can design the perfect part or gizmo for the job at hand. VLSI designers will be able to whip up optimum circuits, with minimum power requirements. Designers of financial software will be able to write the perfect stock market prediction program.

Somewhat intriguingly, this Utopia would have no need for randomness. Randomized algorithms would buy essentially no efficiency gains over deterministic algorithms. (Armchair philosophers should ponder this.)

This Utopia would also come at one price: there would be no privacy in the digital domain. Any encryption scheme would have a trivial decoding algorithm. There would be no digital cash, no PGP, no RSA. We would just have to learn to get along better without these, folks.

We will encounter all these consequences of $\mathbf{P} = \mathbf{NP}$ later in the course.

Exercises

§1 If $\mathbf{P} = \mathbf{NP}$ then there is a polynomial time decision algorithm for 3SAT. Show that in fact if $\mathbf{P} = \mathbf{NP}$ then there is also a polynomial time algorithm that, given any 3CNF formula, produces a satisfying assignment if one exists.

§2 Mathematics can be axiomatized using for example the *Zermelo Frankel* system, which has a finite description. Show that the following language is **NP**-complete.

$$\{< \varphi, 1^n >: \text{ math statement } \varphi \text{ has a proof of size at most } n \text{ in the ZF system}\}.$$

(Hints: Why is this language in **NP**? Is boolean satisfiability a mathmatical statement?) Conclude that if $\mathbf{P} = \mathbf{NP}$ then mathematicians can be replaced by polynomial-time Turing machines.

§3 Can you give a definition of **NEXPTIME** analogous to the definition of **NP** in Definition 2? Why or why not?