COM S 6810 Theory of Computing January 20, 2009

Lecture 1: Introduction

Instructor: Rafael Pass *Scribe: Jean-Baptiste Jeannin*

1 Administration

1.1 Course staff

Professor	Rafael Pass	`rafael@cs.cornell.edu`
TA	Muthu	`vmuthu@cs.cornell.edu`

1.2 Homeworks

There will be 3 to 4 homeworks in the semester. The first homework is already up or will be by the end of the day.

1.3 Textbooks

No textbook is required. A few books have a strong connection with the class:

- Sanjeev Arora and Boaz Barak, *Complexity Theory: A Modern Approach.* To be published around March 2009, but pieces available for free on the internet: `www.cs.princeton.edu/theory/complexity/`
- Dexter Kozen, *Theory of Computation*, Springer, 2006.
- Oded Golreich, *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, 2008.

1.4 Scribing

For each lecture there will be a scribe. The notes have to be sent at most 24 hours after the lecture. Scribing will count in the final grade.

1.5 Project

There will be a project to be done at the end of the semester. This will be explained further later.

2 The Goals of Complexity Theory

What are the goals of complexity theory? First, we want to know what *resources* are needed to solve *important problems*. We also want to define precisely what we mean by *resources*, *important* and *problems*. Second, we want to identify *classes* of problems that *behave* in a *similar way*, i.e., that require similar amounts of resources. These classes will be called complexity classes. Third, we also want to identify relations between these classes.

Example: Let us consider 3 problems: Multiplication, Factoring, SAT:

- what are the best algorithms to solve each problem?
- what relations are there between these problems, i.e., given a solution to one problem, can we know a solution to another one? Here, for example, SAT can solve everything.

To achieve these goals, we need to:

- identify resources (time, space, nondeterministic time, nondeterministic space, etc.).
- place coarse restrictions on them.

But why do we do that? The reasons involved are:

- Pragmatic: Algorithms, Cryptography.
- Philosophical: it allows to define precisely words like *easy* and *hard* ; it also allows us to differentiate a theorem from its proof.

3 Types of problems

Here we will consider solving problems with a Turing Machine. Here are a few computational problems that can be considered:

Decision problem Given $X \in \{0;1\}^*$, decide whether $X \in L$, where $L \subseteq \{0;1\}^*$.

Example: The G3COL (3-color graph) problem. Given a graph, can we color it with only three colors, with the constraint that two vertices incident to an edge shall bear different colors?

In this course, we will focus primarily on decision problems.

Search problems Given $X \in \{0;1\}^*$, find Y such that $(X, Y) \in R$, where $R \subseteq \{0;1\}^* \times \{0;1\}^*$.

Example: In G3COL, find the colors for each nodes satisfying the constraints.

Counting problems The goal here is to determine the number of solutions, for example in G3COL, the number of such colorings. It is easy to see that the counting problem is at least as hard as the corresponding decision problem. Observe that if the number of colorings in G3COL is strictly greater thatn 0 then the graph is indeed 3-colorable.

Optimization problems In general computing the exact solution is hard. Therefore, we try to compute a solution that is nearly as good as the optimal solution. The goal here is to find the best possible solution to some problem.

4 Definitions of P and NP

4.1 P or Polynomial time computation

Fir, we define time bounded computation.

Definition 1 (DTIME) DTIME$(T(n))$ *is the set of all languages* $L \subseteq \{0,1\}^*$ *accepted by a Turing Machine with runtime* $cT(n)$ *on inputs of length* n*, where* c *is a constant not depending on* n*.*

Definition 2 (P) *Then we define:*

$$\mathrm{P} = \bigcup_{c \geq 1} \mathrm{DTIME}(n^c)$$

4.2 NP

Definition 3 (NTIME) *We define* NTIME$(T(n))$ *as the set of all languages* $L \subseteq \{0;1\}^*$ *accepted by a nondeterministic Turing Machine with runtime* $cT(n)$ *on inputs of length* n*, where* c *is a constant not depending on* n*.*

Definition 4 (NP) *Then we define:*

$$\mathrm{NP} = \bigcup_{c \geq 1} \mathrm{NTIME}(n^c)$$

4.3 Other, equivalent NP definition

Definition 5 (NP-search problem) *A relation R is an NP-search problem if R is polynomial time decidable (i.e., can be decided by a polynomial time Turing Machine), and if there exists a polynomial p such that $(x, y) \in R$ only if $|Y| \leq p(|X|)$.*

Definition 6 (NP, alternate) *Then, $L \in$ NP if and only if there exists an NP-search problem R such that $x \in L$ if and only if $\exists y, (x, y) \in R$.*

Equivalence of the two definitions: the idea of the proof is that Y encodes the choices of the nondeterministic Turing Machine.

Remark: If you can decide SAT then you can also find satisfying assignments. Therefore, decision version of SAT is at least as hard as the search version.

5 Reductions and NP-completeness

5.1 Turing reduction

Definition 7 (Turing reduction) *We say that $R_1 \leq R_2$ (R_1 reduces to R_2) if R_1 can be solved using an oracle that solves R_2. The idea is that R_1* is easier than *R_2.*

Using this definition, it can be proven that SAT $\leq$ coSAT, and that NP = coNP.

5.2 Karp reduction

Definition 8 (Karp reduction) *We say that $L_1 \leq_p L_2$ if there exists an* efficient *Turing Machine T such that $x \in L_1$ if and only if $T(x) \in L_2$.*

Belief (but not proven): with this definition, NP$\neq$coNP.

5.3 Meaning of *efficient*

Most of the time *efficient* means polynomial time. We will then talk about polynomial time reductions, or simply *polytime* reductions.

5.4 NP-completeness

We say that L is NP-complete if and only if $L \in \mathrm{NP}$ and $\forall L' \in NP, L' \leq L$.

Theorem 1 (Cook-Levin) 3SAT[1] *is NP-complete.*

It is still not known if P=NP; this is the same as knowing whether we can solve 3SAT in polynomial time.

6 Resources

In this section, we look at the different kinds of resources considered when analyzing complexity.

6.1 Classical resources

These are time, space, nondeterministic time (i.e., time using a nondeterministic Turing Machine), and nondeterministic space.

With these resources, *more of some resource implies more power.* But also, it is very difficult to compare different resources; for example, the P=NP problem is about comparing different resources, and has not been solved yet.

6.2 Alternation

A problem in P can be written as a polynomial Φ such that a time computable predicate $x \in L$ if and only if $\Phi(x)$; a problem in NP can be written as a polynomial predicate Φ such that $x \in L$ if and only if $\exists y, \Phi(x, y)$. The idea of alternation is to add more quantifiers, to get something like:
A problem can be decided with a polynomial Φ such that $x \in L$ if and only if
$\exists y_1, \forall y_2, \exists y_3, ..., \exists y_n, \Phi(x, y_1, ..., y_n)$, for some n.

Playing with quantifiers in such a way gives you more and more power.

6.3 Non uniformity, circuits

We define P/poly to be the set of problems which can be solved in polynomial time using a machine of polynomial size.

[1]Set of all formulas in CNF where each formula has at most 3 variables

6.4 Randomness

Do random coin flips helps in solving problems.

6.5 Counting

In counting, you are supposed to tell how many solutions there are instead of just saying if there are some or not.

6.6 Interaction

In interaction, instead of a written proof like in NP, we give the ability to the machine which decides to ask questions, to interact. IP is the set of problems that can be solved in polynomial time using interaction.

Result: IP=PSPACE

PCP: Reading a few bits of the proof gives you the ability to know whether the proof is correct with some probability; the more bits you read, the higher the probability.

7 Worst case and average case

In cryptography, we are interested in average case hardness rather than worst case hardness. We will also see some worst-case to average-case reductions, where the goal is to make it *as hard on average as in the worst case.*