6.841 Advanced Complexity Theory Feb 7, 2007

Lecture 1

Lecturer: Madhu Sudan *Scribe: Brian Jacokes*

1 Administrative Information

- Lecturer: Madhu Sudan (madhu@mit.edu)
- TA: Swastik Kopparty (swastik@mit.edu)
- Website: `http://theory.csail.mit.edu/~madhu/ST07/`

Make sure you're on the mailing list for the course, if you're already on it you should have received an e-mail on February 1st. The first problem set is already posted and is due in two weeks.

This is an advanced graduate course. It has no text, so we make our own notes instead. As part of that, every student must scribe at least once - this is a requirement to pass the course. Sign up for scribing on the website. Email your scribed notes within 24 hours of the lecture; they do not need to be too polished.

The grading for the class will be based on the following:

- 3 problemsets - First one is out today
- 1 project - Work in pairs, read a paper and present it to the class with any additional progress made. Understanding of the material is essential.
- 1 scribing
- Class participation - Will take place through e-mails between you and the staff

2 Goals of 6.841

6.841 assumes that you know the material presented in 6.840. The first goal of the class will be to identify interesting computational problems. This is obviously a subjective choice, so our definition of "interesting" will be one which is informed by mathematics. For example, a problem may be interesting if many other problems can be reduced to one of its instances, or if it is particularly representative of a larger class of problems.

Once we find an interesting problem, we want to find out how much time and space suffice to solve the problem, and how much are necessary to solve the problem. If we're unable to find an upper or lower bound our alternative is to compare the problem to other existing problems, forming classes and hierarchies of problems.

2.1 Interesting problems

The following three problems are presented as "interesting."

- #SAT ("number-SAT"): Given a 3CNF formula ϕ on n variables $x_1, \ldots, x_n$ with m clauses $c_1, \ldots, c_m$ (so $\phi = c_1 \wedge \ldots \wedge c_m$ and each c_i looks something like $x_{i_1} \vee \bar{x_{i_2}} \vee x_{i_3}$), count the number of satisfying assignments.

- CNF minimization: Given a CNF formula ϕ with m clauses and an integer $m' < m$, does there exist a CNF ψ with at most m' clauses such that $\psi \equiv \phi$? (We say that $\psi \equiv \phi$ if, for each assignment a of $x_1, \ldots, x_n$, we have $\psi(a) = \phi(a)$.)
- Permanent: Given an $n \times n$ matrix $A = [a_{ij}]$ of integers, compute the *permanent* of A. We define $perm(A) = \Sigma_{\pi:[n]\to[n]} \left(\Pi_{i=1}^{n} a_{i\pi(i)}\right)$, where $[n] = \{1, \ldots, n\}$ and π represents a permutation (so $i \neq j$ implies $\pi(i) \neq \pi(j)$). This is basically the formula for the determinant of a matrix, without the power of -1 derived from the sign of each π.

We can now see how these problems fit in with our definition of interesting. To begin with, we can claim that CNF minimization is interesting. CNF minimization arises naturally in circuit design, for instance, if one is able to come up with a straightforward (but large) circuit that solves a problem and then wishes to find a smaller circuit that would achieve the same goal. It is also symptomatic of a large class of problems, which is to say that many other problems look similar or are reducible to CNF minimization. Finally, we may relate CNF minimization to other problems: if P=NP then it turns out that CNF minimization would also be in P, although it is intrinsically different from other satisfiability problems (CNF minimization $\neq$ SAT).

As it turns out, the other two example problems are reducible to each other. The fact that #SAT $\equiv$ Permanent was proved by Valiant, who stated that counting is an interesting "phenomenon." To that end, he created a complexity class called "Algebraic-NP" such that Permanent $\in$ Algebraic-NP-complete. Furthermore, Lipton found that the worst-case instances of Permanent reduce to random instances, implying that the worst-case complexity of the problem is at most the average-case complexity. These problems were also related to CNF minimization by Toda, who found that CNF minimization $\leq$ (*reduces to*) #SAT.

A final note on this topic concerns the paper "The Complexity of Theorem Proving Procedures" by Stephen Cook, which first introduced the concept of NP-completeness. Despite having a great impact on today's complexity theory, the paper was almost rejected when it was first submitted for publication. This demonstrates how recognizing a concept as interesting is far from an exact science and is quite often retrospective, as even one of the most important papers ever released on complexity theory wasn't recognized as interesting at first glance.

3 Technical Definitions

Now we'll give a few precise definitions that will be used throughout the course. To begin with, we'll be dealing with what we call *computational problems.* One definition of a computational problem is that we are given a function f which maps an input x to an output $f(x)$, leading to the problem of finding $f(x)$ given x. A slightly broader definition deals with problems where more than one output is possible for each input: given a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and an input x, produce an output y such that $(x, y) \in R$.

We also consider *languages*, from which *decision problems* arise. A language is defined by $L \subseteq \{0,1\}^*$ and the associated decision problem is that we are given x and wish to decide if $x \in L$. Such problems will be the primary focus of our course for a few main reasons. First of all, it is much easier to find comparisons and reductions between decision problems than between computational problems. For example, it is not clear at all how to reduce an arbitary counting problem to another unless they are sufficiently similar, whereas many seemingly unrelated decision problems often end up being reducible to each other. In addition, many comparison problems have analogous languages, which sometimes allows us to use our findings in decision problems to form broader results about computational problems.

Our principle way of comparing problems will be through *reductions.* We write that $\pi_1 \leq \pi_2$ to mean that the problem π_1 is "no harder than" the problem π_2. This is done by using a reduction from π_1 to π_2, although there is still some freedom as to how we define such a reduction. In the specific setting where π_1 and π_2 are decision problems, characterized by languages L_1 and L_2 respectively, we can use what is called a "Karp reduction" or "many-one reduction." A Karp reduction is written as $L_1 \leq_{Karp} L_2$ and is given by a function $A : \{0,1\}^* \to \{0,1\}^*$ which is both time- and space-efficient to compute, and for which $x \in L_1$ if and only if $A(x) \in L_2$.

The Karp reduction is very specific, so more generally we can use a Turing reduction, which can be thought of as adaptable computation. Such a reduction can solve π_1 by using a subroutine which solves π_2, but also performing computations before, after, and between the subroutine calls. We can see that all Karp reductions also fall into the category of being Turing reductions, but one example of the advantage of Turing reductions over Karp reductions is that they can equate (at least if P=NP) the SAT and $\overline{\text{SAT}}$ problems, whereas this is not possible using a Karp reduction.

A final concept which is used in the course is that of the "complexity zoo." This is generally drawn as a circle with lines demarcating different complexity classes [*Scribe's note*: I'm horribly afraid to try this using my touchpad mouse], and an arrow from one class to another indicating that the first class is no harder than the second class. Examples of complexity classes include L (log-space problems, important in streaming), which is a subset of P (poly-time problems), which is a (proper?) subset of NP (non-deterministic poly-time problems), and so on. Some problems, including the Permanent and CNF minimization problems we discussed earlier, don't have very tight positions within this figure. For instance, the Permanent problem is in PSPACE (poly-space problems) but not in NP or coNP, even though there is a very large gap between these classes of problems.

4 Course Overview

The first three weeks of the course will be spent on provable lower bounds. We won't be spending too much time on this topic because there is an embarassingly small number of good lower bounds on problems.

Although it's easy to prove time and space hierachies (e.g. $\text{TIME}(n^2) \subsetneq \text{TIME}(n^{10})$ or $\text{SPACE}(n^3) \subsetneq \text{SPACE}(n^6)$), it's difficult to compare the two resources. For instance, we know that $\text{TIME}(t) \subseteq \text{SPACE}(t) \subseteq \text{TIME}(2^t)$, but although the first and last classes are not equal we don't know which containment is proper. Likewise we can't prove proper containments in the relations $\text{L} \subseteq \text{P} \subseteq \text{PSPACE}$ or $\text{TIME}(t) \subseteq \text{NTIME}(t) \subseteq \text{TIME}(2^t)$, even though they are there. Along this line, it has been shown that at least one of the following statements is true: SAT $\notin$ Linear-time (which is defined as $\bigcup_{c>0} \text{TIME}(n(\log n)^c)$) or SAT $\notin$ LOGSPACE. A tool which is helpful in this problem is a resource called *alternation*, which will also help us with CNF minimization.

We'll use alternation along with several other resources in our study of complexity theory. Over the course of the term we'll deal with interaction, randomness, proofs, counting problems, and average case complexity. To complete the term we'll investigate quantum computing, which goes against the Turing thesis and is therefore very relevant to our study of computation.