6.841 Advanced Complexity Theory Feb 4, 2009

Lecture 1

Lecturer: Madhu Sudan *Scribe: Mergen Nachin*

1 Administrative Information

- Lecturer: Madhu Sudan (madhu@mit.edu)
- TA: Brenden Juba (bjuba@mit.edu)
- Website: `http://courses.csail.mit.edu/6.841/`

The grading will be based on the following.

- Scribing - You must scribe at least one lecture no matter if you are taking the class for credit or as a listener.
- Problem sets - There will be roughly 3 problem sets throughout the semester.
- Participation - We encourage people to speak up, discuss and ask questions during the lecture.
- Project - Read papers about some topic and present it to the class (with additional progress, if possible).

2 High level overview of Computational Complexity

Computational Complexity is concerned with the study of

- *Interesting* computational problems.
- *Interesting* resources such as time, space and etc.
- The feasibility and infeasibility - That is to prove upper and lower bounds. Unfortunately, we have a very few results on lower bounds for time or space. But on the other hand, we have made quite a progress on *comparison* lower bounds. For example, we compare two problems and conclude the following: if problem A requires some certain amount of resource to solve, then we must need at least some amount of resource to solve problem B.

How do we define "interesting"? This is a very subjective choice. For example, a problem might be interesting if it has a lot of applications in a real world, or if many other problems can be reduced to one of its instances. 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.

2.1 Examples of "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.

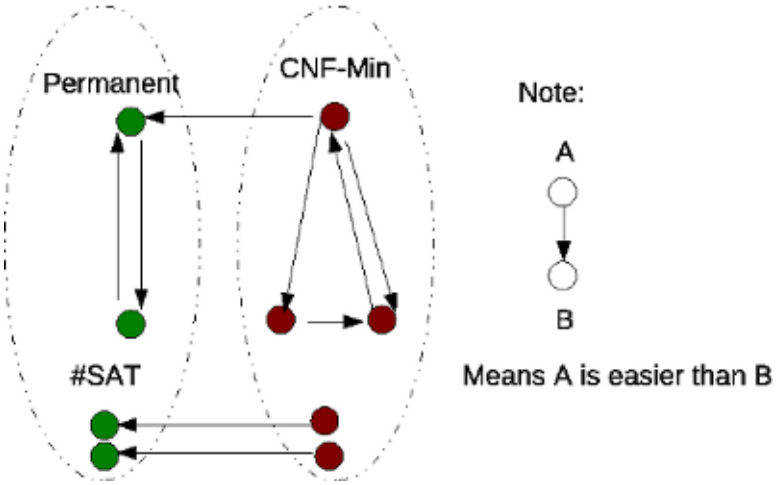

Figure 1: Diagram of relations between problems

- CNF minimization: Given a 3CNF formula ϕ with m clauses and an integer $m' < m$, does there exist a 3CNF 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.

Let's see how these problems fit to our definition of "interesting". CNF minimization is an interesting problem because it has a real world application. For instance, engineers want to optimize a circuit that has as few gates as possible. #SAT is an interesting problem because we encounter it very often. Many types of problems are reducible to #SAT, and is apparently a particular representative of a larger class of problems. As a side note, we don't know which one of CNF-minimization and #SAT is easier, but we will prove some facts about their relation later in the course.

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.

The Permanent problem is a wonderful concept in the mathematical world but unfortunately, we don't know how to compute in $o(2^n)$ time (as opposed to $O(n^3)$ algorithm for computing a determinant). How much space do we need to compute permanent of an n by n matrix? We can solve it in $O(n \log(n))$ space. We also know that we cannot solve it in $o(\log(n))$. But as you probably see, there is a huge gap between the bounds.

Once we relate these problems, we can see that some of them may share some common properties and form an interesting class. For example, we can draw relations between #SAT, CNF-Min, Permanent and many others, and can get the following diagram (see Figure 1). This diagram might suggest that problems

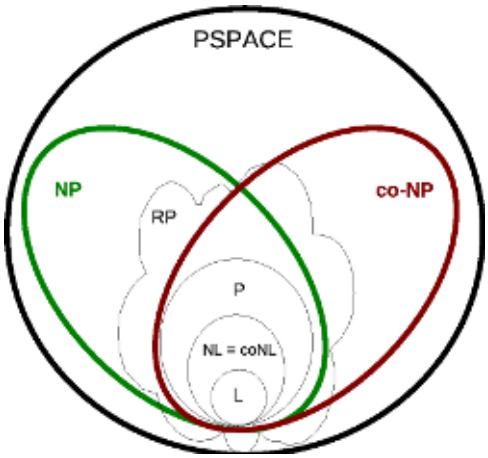

Figure 2: Relationship among L,NL,coNL,P,NP,coNP and PSPACE

labeled as red may form one class, and problems labeled as green may form another class.

From 6.840, we know the following relations among computational classes (see Figure 2). This diagram shows conjectured relations among different classes. We only know that L $\neq$ PSPACE from space hierarchy theorem. Particularly, L might be equal to P or P might be equal to PSPACE but they cannot be equal at the same time. We don't know much about the relationship between RP and NP.

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*. We can define computational problems in three following ways.

- Set or Language L $\in \{0,1\}^*$. Given x $\in \{0,1\}^*$, answer whether x $\in$ L (membership in a set).
- Function. 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.
- Relation. We are given a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and an input x, produce an output y such that $(x,y) \in R$. This is basically definition for searching problems.

Why do we focus on languages (rather than relations)? The reason for focusing only on languages is that there is a simple way of talking about reductions between problems for languages! It is much easier to find comparisons and reductions between decision problems than between search problems.

3.1 Reductions

Reduction allows us to formally compare the relative hardness of different computational problems. There are two main types of reductions in complexity theory:

- **Turing reductions or Many-Many reductions** The most general type of reduction between problems (which works for relations) is of the following type.
  - $R_1 \leq R_2$ (i.e., R_1 reduces to R_2): Given a subroutine to solve R_2, give an algorithm to solve R_1.

- **Karp reductions or Many-One reductions** Reduction between languages can be defined in the following way:
  - A Karp reduction from L_1 to L_2 ($L_1 \leq L_2$) is an algorithm $T : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in L_1 \leftrightarrow T(x) \in L_2$$

We will use the latter kind of reduction in the sequel of this course because Karp reductions are easier to come up (empirically). Note that, there are cases that Turing reductions exist but Karp reductions don't exist (or not found). For instance, consider the language co-SAT=$\{\phi : \phi$ is NOT satisfiable$\}$. SAT is Turing reducible to co-SAT (just call an algorithm to solve SAT as a subroutine and negate the answer). But we believe that SAT is not Karp reducible to co-SAT. This is essentially the same question as proving that a CNF formula does not have a satisfying assignment, where the proof size is polynomial in terms of size of the input.

4 Course overview

The following is roughly what we will do in 6.841.

- Lower Bounds - We will prove some few known lower bounds in terms of time/space ...
- Resources - We will look at another concept called Alternation. This is a new resource, but allows us to get further insights on classical resources such as time/space. For example, consider the possibility (A) that "SAT $\in$ Logspace", and the possibility (B) that "SAT $\in$ LinearTime". We don't know whether they are true individually. We believe (at least, some of us) that neither is true (since both are special cases of the belief that NP $\neq$ P). But using "alternation", we can prove that at most one of (A) and (B) is true.[1]
- New Concepts - such as proofs, interactions, pseudorandom and knowledge.
- Average case complexity vs worst case.
- Quantum computation - Questions concerning about what we can theoretically do in our physical world.

Any suggestions about additional topics are welcome.

From 6.840, you should know the following.

- Time Hierarchy - E.g TIME(n^2) $\subsetneq$ TIME(n^{10}). The proof is by diagonalization method. Remember that, there is a $\log(n)$ slow-down during the simulation.
- Space Hierarchy - SPACE(n^2) $\subsetneq$ SPACE($n^2 \log(\log(n))$). Again, the proof is by diagonalization method.
- TIME(t) $\subseteq$ SPACE(t) $\subseteq$ TIME(2^t). The first containtment is trivial. The second containtment is due to a simulation.
- NTIME(t) $\subseteq$ TIME(2^t). Again by a simulation.
- NSPACE(s) $\subseteq$ SPACE(s^2). Due to Savitch's theorem.
- NSPACE(s) $\subseteq$ coNSPACE(O(s)). By Szelepcsenyi and Immerman (independently).
- NP and NP-Completeness.

[1] Thanks to Michael Forbes for pointing out an error here in an earlier version of these notes.