6.841 Advanced Complexity Theory	February 2, 2005
Lecture 1	
Lecturer: Madhu Sudan	*Scribe: Rafael Pass*

1 Administrivia

The course instructor is Madhu Sudan (madhu@mit.edu), and the TA is Sergey Yekhanin (yekhanin@theory.csail.mit.edu). Send an email to madhu@mit.edu to be added to the course mailing list.

Grading The course will be graded based on 4 problem sets. Each student is also expected to take scribe notes at least once. The first problem set is out now and is due in 2 weeks.

2 Goals of Complexity

The goals of Complexity can be summarized as follows:

- Identify important computational problems,
- Analyze what kind of resources they need to be solved.

Consider, for instance, the following problems: SAT, Sorting, Matchings in graphs, Multiplication, Factoring, Turing machine halting. Natural questions that arise are:

- *What are the best solutions to these problems?*
- *Are they the best possible?*

Secondly, we would like to find out the "relation" between the different problems, by "drawing arrows between them". Towards this goal we first need to define the notion of *efficiency*. This is done by:

1. Identifying a resource of interest,
2. Placing coarse limits on it.

Having defined a notion of efficiency we can now place all the the computational problem of interest as points in a plane and start drawing arrows between them, based on how they relate to each other. More specifically we draw an arrow from A to B (A $\rightarrow$ B) if the following condition holds: *if B can be solved efficiently, then so can A.*

2.1 Computational "Phenomena" - Complexity Classes

Note that if we use a "reasonable" definition of efficiency then the arrows are transitive. We might thus be able to identify, say 2, different sets of problems such that all problems in the sets are connected to each other, but arrows between the sets only go in one direction. Such a computational "phenomena" gives rise to the notion of a *complexity class*. For instance, as far as we know (today) [1] the classes **P** and **NP** have this property. (Also note that the class **co-NP** has the property that only arrows going from the class **P** to **co-NP** are known, while *no* arrows between the classes **NP** and **co-NP** are known. In other words, **co-NP** is "harder" than **P** , but the relation between **NP** and **co-NP** is unknown).

[1] That is, we do not know of any "arrows" going from the class **NP** to **P** .

3 Why is Complexity Interesting?

All the processess in the brain can be seen as a computation. Thus, THOUGHT is Computation. Furthermore, Computation allows us to define many notions that are "natural" to us but "elusive" mathematically. For instance, (as we will see later in the course), Computation allows us to separate notions such as:

1. Easy vs. Hard
2. Proof vs. Theorem
3. Interaction in Proof (debate) vs. Written proof
4. Learning (by yourself) vs. Teaching

In fact, the question if $\mathbf{P} = \mathbf{NP}$ is the *most interesting* mathematical question! Since if $\mathbf{P} = \mathbf{NP}$ there are no "interesting" theorems in mathematics, as they can all be "decided efficiently". If, on the other hand, $\mathbf{P} \neq \mathbf{NP}$, then there is at least one interesting theorem. That is the question $\mathbf{P}$ vs. $\mathbf{NP}$ is "mathematics-Complete" ! (which in turn means that the course 6.841 is MIT-complete).

4 Computational Problems

One way to view a computational problem is as a function on bitstrings x to bitstrings y. This view can be generalized (from functions) to relations, resulting in the following description of a general computational problem:

- *Given a bitstring x, find a bitstring y such that $(x, y) \in R \subseteq \{0,1\}^* \times \{0,1\}^*$.*

In this course we will mainly focus on computational problems for languages. Namely questions of the following kind:

- *Given $x \in \{0,1\}^*$, is $x \in L$?.*

Note that in general, every search problem can be reduced into a decision problem. An example of this will be on the problem set.

Exercise: Reduce the problem of (finding an assignment to) SAT to the language of SAT.

Why will 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!

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

  The problem with this kind of reduction is that it is hard to separate complexity classes using it. For instance, SAT reduces to co-SAT using this reduction. (In fact co-SAT can be solved using a SAT-oracle by simply forwading the instance to the oracle and negating its output).

- **Karp reductions** Reduction between languages can be defined in the following (more stringent) way (this is called a many-one reduction or Karp reduction):
  - A reduction from L_1 to L_2 ($L_1 \to L_2$) 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.

6 Classical vs. Modern Resources in Complexity Theory

The classical resources investigated in Complexity theory are Time, Space and Non-deterministic Time/Space. In this course will will instead focus on the following modern resources:

- **Randomness** (and getting rid of randomness from randomized algorithms). Randomness has played a crucial role in modern Complexity Theory. In recent years the focus has mostly been on removing randomness from algorithms, or reductions. However, although the "derandomized" algorithms no longer make use of randomness, the actual algorithms are very similar to the randomized ones.
- **Non-uniformity**
- **Alternation** Using Alternation we can show that the answer to at least one of the following questions is NO:
  1. SAT $\in$ Lin-Time ?
  2. SAT $\in$ Logspace ?

  (Note that we don't know if the answer to any single of the above questions is no.)
- **Proofs, Interaction** We will, for instance, show that "IP = PSPACE", and present *Probabilistically Checkable Proofs*.
- **Quantum Computation**

7 A 10 Minute Review of 6.840

When you are given more of one resource you can do more. For example,

- TIME(n^2) $\subset$ TIME(n^3) . This is proven using diagonalization techniques. On a high-level, one constructs a language that is in TIME(n^3) and that is different from all languages in TIME(n^2). This is done by simulating all the machine in TIME(n^2) and flipping the output bit on some inputs.
- NTIME(n^2) $\subset$ NTIME(n^3). This was proved by Cook '74 using more complicated techniques.

It seems hard to compare different resources

- *Time/Space*: we have only a very "loose" bound on the relation between space and time, namely
  - TIME(t) $\subseteq$ SPACE(t) $\subseteq$TIME(2^t).
- *Time/NTime* The situation is analogous between deterministic and non-deterministic time, namely
  - TIME(t) $\subseteq$NTIME(t) $\subseteq$ TIME(2^t).

The theory of NP-completeness also investigates the relation between deterministic and non-deterministic time; the main open question being if $\mathbf{NP} = \mathbf{P}$.

- *Space/NSpace* For the case of space vs. non-deterministic space the bounds are nevertheless better,
  - SPACE(s) $\subseteq$NSPACE$\subseteq$SPACE(s^2) (Savitch's theorem)
  - LOGSPACE $\subseteq$ NLOG $\subseteq$ LOG^2
  - NSPACE(s) $\subseteq$ co-NSPACE($O(s)$) (Immerman-Szelepcsnyi)