18.405J/6.841J Advanced Complexity Theory	February 1, 2000

Lecture 1

Lecturer: Daniel A. Spielman Scribe: Shaun Cutts (from 1998: Zulfikar A. Ramzan)

1 Course Overview

1.1 Warnings

Practical warning: this course follows the material in 18.404/6.840, but it is about an order of magnitude harder.

Philosophical warning: much material in the course concerns itself with what would be the case if P $\neq$ NP . If it turns out that P = NP, then much of what you will learn will be not true and/or irrelevant. In addition, the course covers many things that are now considered blind alleys in the attempt to prove that P $\neq$ NP. I think that they are good to learn, so you don't try to do them yourselves. There will be a few things covered of independent interest: e.g. De-randomization.

One other general remark: many of the techniques uses will have an algorithmic flavor. Lately, some practitioners have wondered if there is perhaps too much reliance on this approach.[1]

1.2 Topics to be Covered

- "things about NP"
- randomized classes
- non-uniform classes
- circuit classes & lower-bound proofs
- interactive proofs, probabilisticly checkable proofs (PCPs)
- De-randomization
- possibly quantum complexity

[1] Question: what would be a good example of a different approach? Answer: for instance, in Galois theory, the non-existence of certain things is proved without considering possible constructions in detail.

1.3 Mechanics

There will be 4-5 problem sets. You can collaborate (just cite all collaborators). Please don't look at old solution sets. In addition, one participant will be asked to take scribe notes each class, and write them up in Latex.

2 Review

We won't specify the sort of Turing machine (e.g., one or two tape, or even a RAM or Kolmogorov machine). As long as we are consistent, it won't matter. Different definitions can change times by a polynomial factor, but that does not change the definitions of classes like P and NP. Also note that we used big O notation in our definition. We won't look at constant factors in too much detail for this course. Blum proved results a while back formally showing that we can more or less ignore constant factors for the time complexity classes in which we are interested.[2]

You have all seen definitions for TIME, NTIME, SPACE, and NSPACE. Briefly, for **C** one of the above, $L \in \mathbf{C}(t(n))$ if there exists a (nondeterministic) TM that accepts L in $\{\text{time}, \text{space}\}$ $O(t(n))$.

With the exception of CVP, you will probably be familiar with the following table: (More inclusive classes listed first.)

Class	**Complete Problem**	**Reduction**
EXPTIME	NREGEXP	$\leq_{\mathrm{P}}$
PSPACE	TQBF	$\leq_{\mathrm{P}}$
NP	3SAT	$\leq_{\mathrm{P}}$
P	CVP	$\leq_{\mathrm{L}}$
NL	STCONN	$\leq_{\mathrm{L}}$
L	ISPATH	$\leq_{\mathrm{L}}$

In addition, P $\subset$ BPP $\subset$ PSPACE. P $\neq$ EXPTIME and NL $\neq$ PSPACE are give by the time and space hierarchy theorems:

Theorem 1 (Time Hierarchy Theorem) *If $f(n) = o(g(n)/\lg(n))$, and f, g are "well-behaved", then TIME$(f(n)) \not\subseteq$ TIME$(g(n))$.*

Theorem 2 (Space Hierarchy Theorem) *If $f(n) = o(g(n))$, and f, g are "well-behaved", then SPACE$(f(n)) \not\subseteq$ SPACE$(g(n))$.*

Thus, either L $\neq$ P or $P \neq$ PSPACE (or both). But we haven't been able to prove either.

[2]Basic idea: increase # of states in the machine, and the # of tape symbols, and simulate more than one step at once.

3 P-Completeness and the Circuit Value Problem (CVP)

A language L is said to be *P-complete* under *logspace mapping reductions* if:

1. $L \in P$
2. For all languages $A \in P$, there exists a logspace bounded Turing Machine with a read-only input tape, a write-only output tape, and a read/write work tape, that computes a function f such that: $x \in A \iff f(x) \in L$.

One reason why we are interested in P-complete problems is that we believe they are difficult to compute on a parallel machine. There are many known P-complete problems. We will be interested in the *Circuit Value Problem (CVAL)*. In the Circuit Value Problem, we are given both the description of a circuit and inputs to the circuit, and we are asked to determine the output of the circuit on those particular inputs.[3] It's not hard to see that this can be done in polynomial time. We conjecture, however, that it's hard to come up with an efficient parallel solution.

We will now argue that the CVAL is P-complete under logspace reductions. The following argument is meant to be a proof sketch, and is by no means rigorous. First of all, it's not hard to see that the problem is in P. Suppose we are given $< D, \bar{x} >$, where D is some reasonable description of a circuit (say, a list of circuit types, input and output wires), and $\bar{x}$ is input to that circuit. Then we can simulate the activity of each gate directly by at each step selecting one gate for which all inputs have been calculated, and recording it's output. We can certainly calculate the value of the output wire in polynomial time.

Now, we must show that for every polynomial time bounded TM M, we can construct a logspace computable function f that takes as input $\bar{x}$ and outputs a circuit C and an assignment to the inputs $\bar{y}$ such that $C(\bar{y})$ accepts if and only if M accepts $\bar{x}$ (in polynomial time). Our construction will be similar to the Cook-Levin construction for the NP-completeness of SAT.

Let $t(n)$ be a bound on the running time of M. Assume without loss of generality, that when M is about to accept (reject), it erases its tape, moves its read head all the way to the left, and enters its accept (reject) state. Moreover, M will continue to stay in the same configuration until it has executed $t(n)$ steps. To complete the proof, it is sufficient to specify a logspace TM that outputs a circuit which simulates this $t(n) * t(n)$ tableau. Note that the tableau as a whole is much too big to fit in the memory of our logspace machine. However, it turns out that the structure of the circuit is highly repetitive. All we need to keep in memory is a pointer into the tableau, which is possible in logspace.

[3] We are concerned only with "feed-forward" circuits here: consider a directed acyclic graph with many inputs and one output, where the nodes are typical boolean logic gates. A given input $< D, \bar{x} > \in$ CVP just if D is the description of a circuit and $\bar{x}$ is an appropriately long binary input string such that the output wire of D will output 1 on input $\bar{x}$.

To see this in a little more detail, consider one cell. The information for the cell in the tableau can be represented with a set of wires: One wire for each combination of tape symbol and machine state (representing the state of the machine when the head is not present. We add one "state symbol" to represent the absence of the head.) Only one of this set of wires can be on at any time. The values of the wires in the cell can be determined from the values of the wires in the circuits representing the three cells just above. Other than changes in the numbers to connect wires, the circuitry to simulate the cell is constant. Since all of the numbering needed to output this circuit is local, the machine can operate with a constant number of pointers into the tableau in memory. It completes its task by duplicating the circuitry for a cell $t^2(n)$ times.[4].

4 Final Note

NSPACE($t(n)$) $\subseteq$ SPACE($t^2(n)$) because NSPACE($t(n)$) = co-NSPACE($t(n)$). Proof next time, perhaps.

[4] Of course, we need to fix up this algorithm for the boarders.