6.841/18.405J: Advanced Complexity	**Wednesday, Feburary 5th, 2003**
Lecture 1	
Instructor: Madhu Sudan	*Scribe: Steve Weis*

1 Administrivia

The course instructor is Madhu Sudan, `madhu@mit.edu`. E-mail him if you have not received a class e-mail. The TA is Prahladh Harsha, whose e-mail is `prahladh@theory.lcs.mit.edu` . The course web page is htpp://theory.lcs.mit.edu/~madhu/ST03. Each student is responsible for scribing one lecture. E-mail Prof. Sudan to sign up.

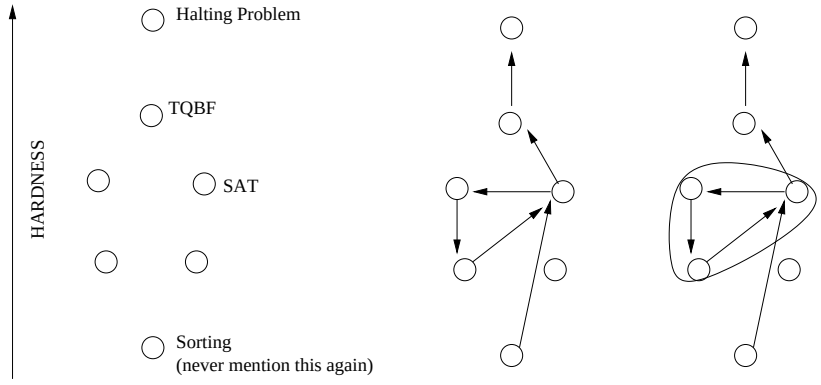
The first homework has been posted on the web page and is due in one week on February 12th. There will be 4-5 problem sets over the course. They will typically be due two weeks after they are assigned.

2 Course Contents

This course examines what kind of computational resources are interesting and how they affect our ability to solve mathematical problems. The 6.840 course dealt with classical concepts of resource: time, space and nondeterminism. This course will consider resources including alternation, non-uniformity, randomness and quantum physics. We will also explore notions of what are proofs, how interaction affects a computation and how knowledge affects computation.

3 Classical Computational Complexity

Traditionally, complexity theory has involved identifying a particular resource, such as time, space or nondeterminism, and quantifying how much of that resource a particular problem requires. By representing a problem as a point on a plane, we can depict its hardness. To indicate that a problem, A, is no harder than another, B, we draw an arrow from A to B. So, $A \to B$ would indicate that the problem A is easier than B, i.e. $A \leq B$. If paths exist in both directions between two problems, then those problems are equivalent difficulty. Clustering sets of equivalent problems yields phenomena such as complexity classes, which we are familiar with. This is depicted below:



4 Complexity Classes

There are many time complexity classes and we would like to be able to find which ones are "interesting" by ruling out as many arrows between them as possible. To start with, we can define classes by applying crude boundaries on resources. These will be logarithmic, polynomial and exponential boundaries on time and space with or without nondeterminism:

	L	NL	P	NP	PSPACE	EXP
Restriction	Space	Space	Time	Time	Space	Time
Boundary	Log	Log	Poly	Poly	Poly	Exp
Nondeterminism	N	Y	N	Y	N	N
Example	Addition	Path	Primes	SAT	TQBF	Chess

Each one of these classes is contained in the class to its right, i.e. $L \subseteq NL \subseteq P \subseteq \ldots, \subseteq EXP$. Note that $PSPACE = NPSPACE$, an important result from 6.840. In fact by Savitch's Theorem, $NSPACE(s(n)) \subseteq SPACE(s^2(n))$ for any $s(n) \geq \log n$. See Section 8.

5 Reductions

Reductions allow for a formal definition of relative hardness between problems. Recall the definition of a language, $L \subseteq \{0,1\}^*$. We are concerned with the computation problem of when given $x \in \{0,1\}^*$, deciding whether $x \in L$. There turn out to be two useful definitions of a reduction:

- Karp Reduction: $L_1 \leq L_2$ if there exists an *efficient* function f such that $x \in L_1 \iff f(x) \in L_2$. This is also called a many-to-one reduction.
- Turing Reduction : $L_1 \leq L_2$ if given access to an oracle deciding L_2, we can decide L_1 *efficiently.* This is also called many-to-many reduction.

Turing reductions are clearly more general since Karp reductions work only between languages while Turing reductions are well-defined for any computational problem. Moreover, a Karp reduction implies a Turing reduction. However, Karp reductions are generally easier to find than Turing reductions. Clearly, a Karp reduction implies a Turing reduction. However, for some problems there are known Turing reductions, but no known Karp reductions. Also, we believe that Karp reductions allow for a finer distinction between classes than what Turing reductions provide us. For example, consider the computational problem $\overline{\text{3SAT}}$ determining whether a 3-CNF formula ϕ is unsatisfiable. Clearly, there exists a Turing reduction $\text{3SAT} \leq \overline{\text{3SAT}}$, that simply negates the output of the oracle. Thus, if we had just Turing reductions, we would be let to believe that $\overline{\text{3SAT}}$ is "no harder" than 3SAT and this would imply that NP = coNP. However Karp reductions allow us to distinguish between NP and coNP as unfortunately, no one has constructed a Karp reduction $\text{3SAT} \leq \overline{\text{3SAT}}$.

6 Hierarchy Theorems

The Time and Space Hierarchy Theories define strict partitions between complexity classes. The theories are analogous to each other, except that the space bound is tighter:

Theorem 1 *For any time constructible function t,* $\text{TIME}(t(n)) \subsetneq \text{TIME}\left(\omega(t(n)\log t(n))\right)$.

Theorem 2 *For any space constructible function s,* $\text{SPACE}(s(n)) \subsetneq \text{SPACE}\left(\omega(s(n))\right)$.

Both of these theorems apply to deterministic classes and are proved through diagonalization arguments covered in 6.840. The basic idea is as follows: Suppose we want to show some language

L is decidable in time $t_2(n)$ but not in time $t_1(n)$. We do this by defining a language L that is decidable in time $t_2(n)$ but different from all languages decidable in $t_1(n)$. L is made to differ from every language in TIME($t_1(n)$) by simulating all the machines that run in time $t_1(n)$ and flipping the answer on some input. Thus, there cannot be any equivalent language in TIME($t_1(n)$), since L explicitly differs from each of them on some input.

Finally, Blum's Speedup Theorem says that any problem solvable in time cn can be solved in time $c'n$ where $0 < c' < c$, i.e.:

Theorem 3 $L \in \text{TIME}(t(n)) \Rightarrow L \in \text{TIME}\left(\frac{t(n)}{2}\right)$

This tells us that the actual constant hidden in the running time of a problem is irrelevant.

7 Open Questions

One of the biggest complexity class questions is whether P=NP. Suppose that P$\neq$ NP, then is it true that we can decide the exact complexity of every problem? For instance, if a language L is described by a non-deterministic Turing machine, then can we decide if $L \in$P? Unfortunately, this problem is undecidable. In fact, under our assumption P$\neq$ NP, there is an infinite chain of problems between P and NP.

Other open problems are whether coNP = NP, L = P or P = PSPACE. Because a turning machine cannot use more space than it has time, TIME($t(n)$) $\subseteq$ SPACE($t(n)$) implying that P $\subseteq$ PSPACE. Similarly, SPACE($s(n)$) $\subseteq$ TIME $\left(2^{O(s(n))}\right)$ which implies that L $\subseteq$ P. However, by the Space Hierarchy theorem, L $\subsetneq$ PSPACE. Thus, we know that one of the containments L$\subset$P, P $\subset$ PSPACE, must be tight.

8 Savitch's Theorem

Savitch's theorem states that something computable in $s(n)$ non-deterministic space is computable in $s(n)^2$ deterministic space. We'll now give a different proof (rather sketch) of this theorem from what is commonly seen in textbooks.

We first use the following claim

Claim 1 *If* $L_1 \leq L_2$ *in* SPACE(S_1) *and* $L_2 \in$ SPACE(S_2), *then* L_1 *is decidable in* SPACE($2S_1+S_2$).

Proof Sketch: This misleadingly appears intuitive. One could first compute the reduction $L_1 \leq L_2$ and then call the decider for L_2. Unfortunately, we are operating under a three-tape Turing machine model: one tape is read-only, one is read/write and the third is write-only. The space restrictions apply only to the read/write tape. The write-only tape, hence the output of the reduction may be of any size. To get around this, we will run the decider for L_2 and then perform the reduction each time the decider needs to read a particular bit of its input. Note that the only space needed for this is $2S_1 + S_2$: S_1, S_2 for the reduction and the decider respectively, while another S_1 is required to indicate which bit is needed.

Now on to Savitch's theorem:

Theorem 4 *For any* $s(n) \geq \log n$, NSPACE($s(n)$) $\subset$ SPACE($s^2(n)$)

Proof Sketch: We did not reach this in class, however, we'll provide a brief sketch below.

We'll use Claim 1 repeatedly to prove this Theorem. Let $N = 2^{s(n)}$. Consider the language

$$\text{PATH}_{N,i} = \{\langle G, s, t\rangle : |V(G)| = N, \text{ there exists a path of length at most } i \text{ from } s \text{ to } t.\}$$

Since $\text{PATH}_{N,N}$ is a complete problem for NSPACE($s(n)$), it is sufficient if we show that $\text{PATH}_{N,N} \in$ SPACE($s^2(n)$). For this, we reduce $\text{PATH}_{N,i}$ to $\text{PATH}_{N,i/2}$ in the following manner and then use

Claim 1. From an instance $\langle G, s, t\rangle$ of $\text{PATH}_{N,i}$, we construct the new graph G' which has exactly the same set of vertices in G but two vertices in G' are connected if there is a path of length at most 2 in the original graph G. Thus, $\langle G, s, t\rangle \in \text{PATH}_{N,i} \iff \langle G', s, t\rangle \in \text{PATH}_{N,i/2}$. Clearly this reduction takes space $O(\log N)$. Suppose, S_i is the space required to decide $\text{PATH}_{N,i}$, then by Claim 1, we have that recurrence $S_i \leq S_{i/2} + O(\log N)$. Also, we note that $S_1 = O(1)$. Solving the recurrence we have $S_i = (\log i)(\log N)$. Hence, $S_N = (\log N)^2 = s^2(n)$, which is what we wanted to prove.