18.405J/6.841J Advanced Complexity Theory	February 22, 2001

Lecture 5

Lecturer: Dan Spielman — *Scribe: Adam Winkel*

In this lecture, we will prove a major theorem in Complexity Theory which states that if NP is contained in P/poly, then the polynomial hierarchy collapses. Because it is currently thought that the polynomial hierarchy is incollapsable, this leads us to believe that NP problems cannot be solved with polynomial sized circuits. The proof of the theorem will illustrate some interesting techniques in Complexity Theory.

Toward the end of the lecture, we will consider some general results in circuit complexity starting with Claude Shannon's initial result on the number of hard functions.

1 NP and P/poly

First, recall the definition of P/poly :

$$\text{P/poly} = \{ \text{ all languages, } L, \text{ where } \exists k > 0, A \in \text{P}, \text{ and a sequence of strings } \{s_n\}_{n\in\mathcal{N}} \text{ such that } |s_n| = O(n^k) \text{ and } w \in L \iff (w, s_{|w|}) \in A\}$$

The s_n strings can be thought of as input-size dependent advice given to a Turing machine. We will start by assuming that the canonical NP-complete problem, SAT $\in$ P/poly, and then define what it means to for a sequence of strings to be a *good* sequence of advice strings for SAT. In turn, we will define the language, GOOD, of good advice sequences, and then we will show that GOOD $\in$ coNP. This allows us to construct a $\Sigma_2 P$ TM that can guess a *good* sequence, then verify that the sequence is *good*, and finally use the sequence to decide complete problems for $\Sigma_3 P$. Therein lies the collapse of PH.

Theorem 1 (Karp, Lipton, Sipser) *If* NP $\subset$ *P/poly then* $\Sigma_3 P \equiv \Sigma_2 P$ *(PH collapses)*

Take the NP complete problem SAT. Since SAT $\in$ P/poly, then there is some constant k, some polynomial language, $A \in P$, and a sequence of strings $\{s_n\}_{n\in\mathcal{N}}$ such that $|s_n| < kn^k$ and $w \in SAT \iff (w, s_{|w|}) \in A$. We fix the language A for the rest of this lecture.

Definition 2 *The sequence of strings* $(g_1, ..., g_l)$ *is a* good *sequence if*

1. $\forall i,\ |g_i| \leq ki^k$
2. $\forall \phi,\ |\phi| < l, \phi \in SAT \iff (\phi, g_{|\phi|}) \in A$

Definition 3 GOOD $= \{(g_1, ..., g_l) | (g_1, ..., g_l)$ *is a good sequence* $\}$

It is easy to see that GOOD $\in \Pi_2 P$ by exploiting an earlier definition that $\Pi_2 P =$ co-Np^{SAT}. Consider the following ATM:

1. On input $(g_1, ..., g_l)$, first check to make sure that $|g_i| \leq ki^k$ for $1 \leq i \leq l$. Otherwise, reject.
2. Next, use the $\forall$ states to choose a boolean formula ϕ such that $|\phi| < l$.
3. Let a be a boolean variable. Use the polynomial TM to compute $a \leftarrow (\phi, s_{|\phi|}) \overset{?}{\in} A$.
4. Ask the SAT oracle whether $\phi \in SAT$.

5. Accept if $a \iff \phi \in SAT$. Otherwise, reject.

However, we can prove an even better containment, GOOD $\in$ coNP, by understanding the notion of self-reducibility.

Definition 4 *A language L is self-reducible if there exists a polynomial time Oracle Turing Machine (OTM) M such that*

1. *M^L accepts L*
2. *on inputs of length n, M only queries the oracle about words of length less than n.*

One can think about self-reducible languages as those accepted by some sort of dynamic programming Turing machine that can look up answers to smaller problems. Now we show that SAT is self-reducible.

Lemma 5 *SAT is self-reducible.*

Proof Consider a boolean formula ϕ of n variables, $\phi(x_1, ..., x_m)$. We can fix $x_1 = TRUE$ and $x_1 = FALSE$ and evaluate the formula ϕ with those assignments. The resulting formulas

$$\begin{aligned} \phi_0 &= \phi(0, x_2, ..., x_m) \\ \phi_1 &= \phi(1, x_2, ..., x_m) \end{aligned}$$

will be smaller formulas, and moreover, $\phi \in SAT$ iff ($\phi_0 \in SAT \vee \phi_1 \in SAT$) $\blacksquare$

Lemma 6 GOOD $\in \Pi_1 P$ (*i.e.* coNP)

Proof Consider the following Turing machine that decides GOOD:

1. On input $(g_1, ..., g_l)$, check the length requirement as before.
2. As before, use $\forall$ states to choose ϕ such that $|\phi| \leq l$.
3. As before, $a \leftarrow (\phi, s_{|\phi|}) \stackrel{?}{\in} A$.
4. If ϕ contains no bound variables (base case), then evaluate ϕ and accept only if $a \iff \phi \in SAT$.
5. Otherwise, use self-reduction to construct ϕ_0, ϕ_1.
6. Let $a_i \leftarrow ((\phi_i, g_{|\phi_i|}) \in A)$.
7. Accept iff $a \iff a_0 \vee a_1$.

Note that we only use $\forall$ states, so this TM is in coNP.

We show correctness by first observing that we could either 1) reject a *good* sequence, or 2) accept a non-*good* sequence. It is easy to see that we will accept all *good* sequences.

Now we show that we will reject all non-*good* sequences. Observe that sequences can fail to be *good* in two ways:

1. A sequence might cause A to reject an otherwise satisfiable formula. In this case, at step (3), the boolean variable a will be $false$. If ϕ is trivial, then we will evaluate it and notice that it in fact is $\in SAT$. Therefore, our machine will reject the sequence.

   If ϕ is not trivial, then we will compute a_i in step (6). Observe that since the sequence $(g_1, ..., g_l)$ is a bad sequence, then there is some minimal length satisfiable formula $\phi_m in$ on which $(g_1, ..., g_l)$ convinces A to reject. We can assume that our $\forall$ states will find this θ. However, since $\phi_{min_0}, \phi_{min_1}$ are smaller formulas, then $(g_1, ..., g_l)$ will offer the correct advice on these strings. In particular, since $\phi_m in$ was satisfiable, either $a_0 =$ TRUE or $a_1 =$ TRUE. Therefore, our machine will reject in step (7).

2. The sequence might convince A to accept a formula that is not satisfiable. At step (3), the boolean variable a will be set to $true$. Like before, the base case in (4) will reject the sequence immediately if ϕ evaluates to false.

   Otherwise, we will compute a_i as before. The same argument about minimal $\phi_m in$ applies in this case. There is some minimal length unsatisfiable formula $\phi_m in$ on which $(g_1, ..., g_l)$ convinces A to accept. Since ϕ_{min_i} are smaller formulas, then the sequence will advise A to reject both of them since neither are satisfiable. Hence, our machine will reject in step (7).

The key idea is to understand that if the sequence *ever* deceives us, then there must be some minimal length formula on which it deceives us. Our $\forall$ states are guaranteed to find this minimal length case. By using self-reduction, this minimal length case can be transformed into two smaller questions on which the sequence must act correctly. In this way, we can detect the deception and reject. Remember that one rejecting branch of a coNP computation tree suffices to reject altogether. ■

Now we return to our original goal, proving the theorem of Karp, Lipton , and Sipser. To refresh our memory, we restate the theorem.

Theorem 7 (Karp, Lipton, Sipser) *If* NP $\subset$ *P/poly, then* $\Sigma_3 P \equiv \Sigma_2 P$ *(PH collapses)*

Proof We have already shown that $SAT \in$ P/poly and that GOOD $\in$ coNP.
Let L be a language in $\Sigma_3 P$. Thus, $\exists B \in P$ and a polynomial p(.), such that

$$w \in L \iff (\exists x \forall y \exists z (w, x, y, z) \in B and |x|, |y|, |z| \leq p(w).$$

To complete the proof of the theorem, we present a Turing machine that decides L. On input w:

1. Use $\exists$ states to guess an x and a *good* sequence of strings $(g_1, ..., g_l)$ for l "sufficiently large," to be determined later.

2. Use $\forall$ states to guess y. Continue to use $\forall$ states to test if $(g_1, ... g_l) \in$ GOOD using the above lemma. If not, we reject.

3. We want to know if $\exists z$ such that $(w, x, y, z) \in B$. This problem is in NP.

4. Reduce the above problem to an NP-complete problem: SAT. Construct ϕ such that $\phi \in SAT \iff \exists z$ such that $(w, x, y, z) \in B$.

5. Use the good sequence $(g_1, ..., g_l)$ to decide whether $(\phi, g_{|\phi|}) \in A$. Accept if A accepts, else reject.

From the last step, we can see that l must be $\geq |\phi$. The machine alternates between $\exists$ and $\forall$ states only once, and so it is in $\Sigma_2 P$. ■

2 Circuit Complexity

Theorem 8 (Claude Shannon) *There is a sequence of functions* $f : \{0.1\}^n \rightarrow \{0,1\}$*, such that the circuit complexity,* $C(f) = \Omega(\frac{2^n}{n})$.

Aside: It is not too hard to show that $\forall f, C(f) \leq O(2^n)$

Proof This is a simple counting argument. There are 2^{2^n} functions from $\{0,1\}^n \rightarrow \{0,1\}$. To show this, consider each function as a table of ones and zeroes for each n-bit number. Hence, each function can be represented as a 2^n bit string, and there are a total of 2^{2^n} different 2^n bit strings.

Consider the number of circuits that consist of g and/or/not gates. There are 3^g different sets of gates we can use, and there are $< (g+n)^{2g}$ different input configurations for the gates. Hence, there are at most $3^g(g+n)^{2g}g$ different boolean circuits of g gates. Consider the set of functions such that $C(f) < g$. This implies that $2^{2^n} < 3^g(g+n)^{2g}$. Taking logs, we have $2^n < 2g(\log(g+n+3/2)) + \log g$ which implies that $g > \frac{2^n}{2n}$. There are some more complicated methods to eliminate the 2 in the denominator. ■