18.405J/6.841J Advanced Complexity Theory	March 12, 1998
Lecture 10	
Lecturer: Daniel A. Spielman	*Scribe: Joshua A. Tauber*

Valiant and V. J. Vazirani (cont.)

Consider a set $S \subseteq \{0,1\}^N$, random vectors $\vec{v_0}, \ldots \vec{v_N}$, and the series of subsets:

$$\begin{aligned}
\sigma_1 &= \{\sigma \in S \mid \vec{v_0} \cdot \sigma = 0\} \\
\sigma_2 &= \{\sigma \in S \mid (\vec{v_0} \cdot \sigma = 0) \wedge (\vec{v_1} \cdot \sigma = 0)\} \\
&\vdots \\
\sigma_N &= \{\sigma \in S \mid (\vec{v_0} \cdot \sigma = 0) \wedge (\vec{v_1} \cdot \sigma = 0) \wedge \ldots \wedge (\vec{v_N} \cdot \sigma = 0)\}
\end{aligned}$$

where all operations are mod 2. Last time we proved:

Fact 1 *If* $|S| \geq 1$ *then* $Pr[\exists! i \text{ such that } |\sigma_i| = 1] \geq \frac{1}{8}$

Let ϕ be a boolean formula in 3-CNF form over N boolean variables $(x_1, x_2, \ldots, x_N)$. We may elect to treat these variables as a vector, $\vec{X}$. Choose $N+1$ random vectors of length N : $\vec{v_0}, \vec{v_1}, \ldots, \vec{v_N}$. We define

$$\begin{aligned}
\phi_0 &= \phi \wedge (\vec{v_0} \cdot \vec{X} = 0) \\
\phi_1 &= \phi \wedge (\vec{v_0} \cdot \vec{X} = 0) \wedge (\vec{v_1} \cdot \vec{X} = 0) \\
&\vdots \\
\phi_N &= \phi \wedge (\vec{v_0} \cdot \vec{X} = 0) \wedge (\vec{v_1} \cdot \vec{X} = 0) \wedge \ldots \wedge (\vec{v_N} \cdot \vec{X} = 0)
\end{aligned}$$

Observation 2 *Using Fact 1 we observe that:*

- *If* ϕ *is* not *satisfiable then neither are any of* $\phi_0, \phi_1, \ldots, \phi_N$.
- *If* ϕ is *satisfiable* $Pr[\exists i \mid \phi_i$ has exactly one statisfying assignment$] \geq \frac{1}{8}$

Claim 3 ϕ_i *can be transformed into a 3-CNF that is true IFF* ϕ_i *is true.*

Now we have a randomized production from $3-SAT$ to N instances of $3-SAT$ such that

- If the original $3-SAT$ instance is false then all N are false.
- If the original $3-SAT$ instance is satisfiable then

$$Pr[\text{One of these instances is satisfied by exactly one assignment}] \geq \frac{1}{8}$$

Finally, we state our findings in the form we find most interesting:

Theorem 4 *If there exists a randomized, polynomial time Turing machine, M, such that on input w*

- *M rejects w if w is a boolean formula in 3-CNF that is* not *satisfiable.*
- *if w is a boolean formula in 3-CNF that has exactly one satisfying assignment, then* $Pr[M \text{ accepts } w] \geq 1/2$.
- *M may accept or reject* arbitrarily *otherwise.*

then $NP = RP$.

Observation 5 *M accepts* $w \Rightarrow w \in SAT$. *In other words, M has zero-sided error. That is the key to getting a language in RP.*

Proof

$\supseteq$ Obvious.

$\subseteq$ Given M, we construct another randomized polynomial time TM, M', such that $L(M') = 3-SAT$. On input w, M' uses random bits to construct $N+1$ new instances of $3-SAT$ (as in Claim 3) and runs M on all $N+1$ instances. M' accepts if M accepts any of them. If $w \in 3-SAT$ then $Pr\left[M' \text{ accepts some instance}\right] \geq \frac{1}{16}$. If not then $Pr\left[M' \text{ rejects}\right] = 1$. Thus, $L(M')$ is RP version of $3-SAT$.

■

Corollary 6 *The existence of M implies collapse of the polynomial hierarchy.*

Proof We know that $RP \subseteq P/poly$. Therefore, $NP = RP \Rightarrow NP \subseteq P/poly \Rightarrow \Sigma_2 P = \Pi_2 P$ ■

Randomized Boolean Polynomials

The previous result says something fundamental about the OR ($\bigvee$) function. Somehow, the addition of random bits makes $\bigvee$ equivalent to $\exists!$. Next we pursue this idea further by considering boolean formulas in the form of randomized polynomials over the real numbers, $\mathbf{R}$. (Or polynomials interpreted as boolean formulas.)

Let

$$\begin{aligned} TRUE &\equiv 1 \\ FALSE &\equiv 0 \\ \bigwedge(x_1, \ldots, x_N) &\equiv \sum_{i=1}^{N} x_i \\ \bigvee(x_1, \ldots, x_N) &\equiv 1 - \sum_{i=1}^{N}(1 - x_i) \\ \neg(x) &\equiv 1 - x \end{aligned}$$

Degree of non-randomized boolean polynomials

Notice that the AND and OR polynomials given have degree N.

Claim 7 *Any polynomial function for AND over $\{0,1\}^N$ will have degree at least N. .*

Proof Given that $x \in \{0,1\}$, no variable will occur with degree > 1 since $x = x^2$. Furthermore, AND functions of subsets of xi's form a basis of all functions over $\{0,1\}^N$. Consider the matrix of AND functions of subsets of $\{0,1\}^N$.

$$
\begin{array}{cc}
 & \overbrace{\begin{array}{ccccccc} \{x_1\} & \ldots & \{x_N\} & \{x_1,x_2\} & \ldots & \{x_1,x_N\} & \ldots \; \{x_1,\ldots,x_N\} \end{array}}^{\text{Index with all members of the power set of } \{0,1\}^N} \\
\begin{array}{c}\text{Index with}\\ \text{all possible}\\ \text{assignments}\\ \text{to } \{0,1\}^N\end{array} \left\{ \begin{array}{c} \{0,\ldots,0\} \\ \{0,\ldots,1\} \\ \vdots \\ \{1,\ldots,1\} \end{array} \right. & \left(\begin{array}{c} \\ \text{Value of entry } (i,j) = AND(j^{th} subset) \text{ with } i^{th} \; assignment. \\ \\ \\ \end{array} \right)
\end{array}
$$

We leave it as an exercise to the reader to show that the matrix thus formed (with proper ordering of the indices) is an upper triangular matrix. Thus, the AND functions are linearly independent, forming the needed basis. ■

A Randomized Polynomial for Approximating OR

While we cannot find a lower degree deterministic polynomial for AND (or OR), we can do better with the addition of random bits. We will use Fact 1 show how to make a randomized polynomial $P(x_1, \ldots, x_N, r_1, \ldots, r_M)$ where $x_i, r_i \in \{0,1\}$ such that

- P has degree $\mathcal{O}(log^2 N)$.
- P uses $\mathcal{O}(log^2 N)$ random bits.
- If $\bigvee_{i=1}^{N} x_i = 0$ then for all possible assignments to $(r_1, \ldots, r_M), P(x_1, \ldots, x_N, r_1, \ldots, r_M) = 0$.
- If $\bigvee_{i=1}^{N} x_i = 1$ then for at least $\frac{1}{8}$ of possible assignments to $(r_1, \ldots, r_M), P(x_1, \ldots, x_N, r_1, \ldots, r_M) = 1$.
- Otherwise, the value of $P(x_1, \ldots, x_N, r_1, \ldots, r_M)$ is arbitrary and, in particular need not be $\in \{0,1\}$.

Let $n = \lceil \log N \rceil$. We will write index $1 \leq i \leq N$ in binary as $i_1 i_2 \ldots i_j \ldots i_n$. That is, we can treat each index i as a binary vector of length n. Similarly, $(r_i, \ldots, r_M)$ will be interpreted as giving $n+1$ binary vectors of length n. Thus, $M = n(n+1)$.

Note: If at most one of $x_1, \ldots, x_N = 1$ then $\bigvee_{i=1}^{N}(x_i) = \sum_{i=1}^{N}(x_i)$.

Parity Polynomial

As a step to writing P, we define a polynomial that computes the inner product of index i (taken as a vector) and a corresponding vector of random bits. Since all operations are mod 2, this calculation reduces to simply computing the parity of the individual terms of the inner product.

We simplify further by transforming our basis so that $TRUE \equiv -1$ and $FALSE \equiv 1$. Under this basis, parity is simply the product of terms. The transformation is a simple linear one:

$$FALSE \equiv 0 \overset{y=1-2x}{\longrightarrow} FALSE \equiv 1$$
$$TRUE \equiv 1 \overset{y=1-2x}{\longrightarrow} TRUE \equiv -1$$

(The inverse of this transformation is $x = \frac{1}{2}(1-y)$.) So, we define the parity polynomial E:

$$\begin{aligned}
E(i_1,\ldots,i_n,r_1,\ldots,r_n) &= \vec{\imath}\cdot\vec{r} \bmod 2 \\
&= Parity(i_1r_1, i_2r_2, \ldots, i_nr_n) \\
&= \prod_{j=1}^{n}(1-2i_jr_j)
\end{aligned}$$

Transforming E back to our original basis yields:

$$E = \frac{1}{2}\left[1 - \prod_{j=1}^{n}(1-2i_jr_j)\right]$$

As it turns out, we are more interested in $\overline{E} = 1 - E$. Multiplying x_i by $\overline{E}$ can be used to select x_i such that $\vec{\imath}\cdot\vec{r} = 0$. Thus,

$$\begin{aligned}
\sum_{\vec{\imath}\cdot(r_1\ldots r_n)=0} x_i &= \sum_{i=1}^{N} x_i\overline{E}_i \\
&= \sum_{i=1}^{N} x_i\left[1 - \frac{1}{2}\left(1 - \prod_{j=1}^{n}(1-2i_jr_j)\right)\right]
\end{aligned}$$

S_a Polynomial

We can now use $\overline{E}$ to construct P. But first, for ease of notation, lets renumber P's random bits, r_i, as $n+1$ vectors.

$$\begin{pmatrix} r_1 & \ldots & r_n \\ r_{n+1} & \ldots & r_{2n} \\ & \vdots & \\ r_{n^2+1} & \ldots & r_M \end{pmatrix} \longrightarrow \begin{pmatrix} r_1^1 & \ldots & r_n^1 \\ r_1^2 & \ldots & r_n^2 \\ & \vdots & \\ r_1^{n+1} & \ldots & r_n^{n+1} \end{pmatrix} = \begin{pmatrix} \vec{r}^1 \\ \vec{r}^2 \\ \vdots \\ \vec{r}^{n+1} \end{pmatrix}$$

Using this renumbering with the parity polynomial yields:

$$\begin{aligned}
\vec{\imath}\cdot\vec{r}^k = 0 &\Leftrightarrow \overline{E}_{i,k} = 1 \\
(\vec{\imath}\cdot\vec{r}^1 = 0) \wedge (\vec{\imath}\cdot\vec{r}^2 = 0) \wedge \ldots \wedge (\vec{\imath}\cdot\vec{r}^a = 0) &\Leftrightarrow \prod_{k=1}^{a}\overline{E}_{i,k} = 1
\end{aligned}$$

We now define polynomial S_a.

$$\begin{aligned}
\underset{1\le a\le n+1}{S_a} &= \sum_{i=1}^{N}\left[x_i\prod_{k=1}^{a}\overline{E}_{i,k}\right] \\
&= \sum_{i=1}^{N}\left[x_i\prod_{k=1}^{a}\left[1-\frac{1}{2}\left[1-\prod_{j=1}^{n}(1-2i_jr_j^k)\right]\right]\right]
\end{aligned}$$

We know that $\bigvee_{i=1}^{N}(x_i) = 0 \Rightarrow S_a = 0$. Otherwise, $Pr[\text{some } \mathrm{S_a} = 1] \geq \frac{1}{8}$ over the choice of random bits. One way to look at S_a is that S_a is the number of true valued x_i that remain after restricting the OR with a random, linear restrictions.

[We will finish next time . . .]