Lecture 14

PCP and NP

May 20, 2004
Lecturer: Paul Beame
Notes: Terri Moore

Last time we finished the proof of Babai, Fortnow, and Lund's theorem that $\mathsf{PCP}(\mathsf{poly}, \mathsf{poly}) = \mathsf{NEXP}$. The following is an almost immediate corollary based on scaling down the parameters in the proof to NP instead of NEXP.

Corollary 14.1. $\mathsf{NP} \subseteq \mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$.

We will prove the following somewhat stronger theorem of Babai, Fortnow, Levin, and Szegedy that yields polynomial-size proofs and a theorem by Feige, Goldwasser, Lovasz, Safra, and Szegedy that does not yield polynomial-size proofs but has much better parameters.

Theorem 14.2 (Babai-Fortnow-Levin-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$*; moreover the proof table for the* $\mathsf{PCP}(\mathsf{polylog}, \mathsf{polylog})$ *algorithm is of polynomial size.*

Theorem 14.3 (Feige-Goldwasser-Lovasz-Safra-Szegedy). $\mathsf{NP} \subseteq \mathsf{PCP}(\log n \log\log n, \log n \log\log n)$.

We will go through one proof that will yield both theorems. Before doing so, we will prove a connection between PCPs for NP and MAX-CLIQUE.

Definition 14.1. For an undirected graph G, MAX-CLIQUE(G) (also known as $\omega(G)$) is the size of the largest clique in G. An function f is a factor α approximation algorithm for MAX-CLIQUE iff MAX-CLIQUE$(G)/\alpha \leq f(G) \leq$ MAX-CLIQUE(G).

Theorem 14.4 (Feige-Goldwasser-Lovasz-Safra-Szegedy). *If* $\mathsf{NP} \subseteq \mathsf{PCP}(r(n), q(n))$ *and there is a polynomial time algorithm approximating* MAX-CLIQUE *better than a factor of* 2 *then* $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{O(r(n)+q(n))})$.

Proof. Let $L \in \mathsf{NP}$ and $V_L^?$ be a polytime verifier in a $\mathsf{PCP}(r(n), q(n))$ protocal for L.

A *transcript of* $V_L^?$ can be described by a random string $r \in \{0,1\}^{r'(n)}$ where $r'(n)$ is $O(r(n))$ and the pairs (q_i, a_i) of queries and answers from the oracle for $i = 1, \ldots, q'(n)$, where $q'(n)$ is $O(q(n))$, and $a_i \in \{0,1\}$.

On input x, transcript t is *accepting* if and only if $V_L^?(x, r)$ given oracle answers $(q_1, a_1), \ldots, (q_{|x|}, a_{|x|})$ accepts. Two transcripts $t = (r, q, a)$, $t' = (r', q', a')$ are *consistent* if and only if $\forall i, j$ if $q_i = q_j$ then $a_i = a'_j$.

There are $2^{r'(n)+q'(n)}$ total transcripts on an input x with $|x| = n$ since the queries q_i are determined by the random string and the previous a_i's. Define a graph G_x with

$$V(G_x) = \{ \text{ accepting transcripts of } V_L^? \text{ on input } x\}.$$

In G_x, $t, t' \in V(G_x)$ are connected by an edge if and only if t, t' are consistent accepting transcripts. Since $V_L^?$ is polynomial time, we can verify whether a transcript is in $V(G_x)$ and check any edge of $E(G_x)$ in polynomial time. (Note: There is no need to write out the q_i part in creating the graph since the q_i can be determined as above.)

Observe that a clique in G_x corresponds precisely to all accepting computations based on a single oracle. In one direction, if an oracle is fixed then all accepting computations given that oracle will have consistent transcripts. In the other direction, for a clique in G_x, any oracle query yields the same answer on all the transcripts in the clique and therefore we can extend those answers consistently to a single oracle for which the transcripts in the clique correspond to accepting computations on that oracle.

Therefore

$$\max_{\Pi} \Pr[V_L^{\Pi}(x, r) \text{ accepts }] = \text{MAX-CLIQUE}(G_x)/2^{r'(n)}.$$

(As a sanity check, notice that there are at most $2^{r'(n)}$ mutually consistent transcripts; otherwise there would be two consistent transcripts with the same random strings and different query answers and these must be inconsistent with each other.)

Therefore by the PCP definition, if $x \in L$ then G_x has a clique of size $2^{r'(n)}$ but if $x \notin L$ then G_x has a clique of size at most $2^{r'(n)}/2$. The algorithm for L simply runs the approximation algorithm on input G_x and accepts if the answer is larger than $2^{r'(n)}/2$. The running time is polynomial is the size of G_x which is $2^{O(r(n)+q(n))}$. □

Proof of Theorems 14.2 and 14.3. The proof will follow similar lines to that of Babai, Fortnow, and Lund. Given a 3-CNF formula φ we can express φ implicitly as a formula in fewer variables just like we did using the B formula. The following table summarizes the similarities and represents the scaled down parameters.

Before [BFL]	Now
2^n vars indexed by n bits	n vars indexed by $\log n$ bits
2^m clauses indexed by m bits	$\leq 8n^3$ 3-clauses indexed by $\ell = 3 \log n + 3$ bits
$A : \{0,1\}^n \to \{0,1\}$	$a : \{0,1\}^{\log n} \to \{0,1\}$
$\lvert\mathbb{F}\rvert$ polynomial in n	$\lvert\mathbb{F}\rvert$ polylog in n

In order to understand things better we will now express the proof table explicitly and use the sum-check protocol for the verification because it is more explicit. Let $i_1, i_2, i_3 \in \{1, \ldots, n\}$ be indices of clauses. (We use i instead of v to emphasize their size.) We can view the 3-CNF formula φ as a map $\widehat{\varphi} : \{0,1\}^\ell \to \{0,1\}$ saying which of the $8n^3$ possible clauses appear in φ. That is,

$$\widehat{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) = 1 \iff \text{the clause denoted } (x_{i_1} = s_1) \vee (x_{i_2} = s_2) \vee (x_{i_3} = s_3) \text{ is in } \varphi.$$

φ is satisfiable iff there is an assignment $a : \{0,1\}^{\log n} \to \{0,1\}$ such that

$$\forall (i_1, i_2, i_3, s_1, s_2, s_3) \in \{0,1\}^\ell (\widehat{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) = 0 \text{ or } a(i_1) = s_1,\ a(i_2) = s_2, \text{ or } a(i_3) = s_3).$$

Let $\bar{a}$ and $\bar{\varphi}$ be multilinear extensions of a and $\widehat{\varphi}$. Then

$$\varphi \text{ is satisfiable} \iff \forall (i_1, i_2, i_3, s_1, s_2, s_3) \in \{0,1\}^\ell\ \bar{\varphi}(i_1, i_2, i_3, s_1, s_2, s_3) \cdot (\bar{a}(i_1) - s_1) \cdot (\bar{a}(i_2) - s_2) \cdot (\bar{a}(i_3) - s_3) = 0.$$

In polynomial time the verifier can easily produce $\bar{\varphi}$ on its own. The verifier will run a multilinearity test on $\bar{a}$ as before. Let $y = (i_1, i_2, i_3, s_1, s_2, s_3)$ and for any function a let

$$CC(y, a) = \bar{\varphi}(y) \cdot (a(i_1) - s_1) \cdot (a(i_2) - s_2) \cdot (a(i_3) - s_3)$$

be the correctness check for clause y and assignment a. Thus φ is satisfiable if and only if $\forall y \in \{0,1\}^\ell\ CC(y,\bar{a}) = 0$. Observe that the verifier can efficient produce the polynomial for CC and since $\bar{\varphi}$ is multilinear, if $\bar{a}$ is multilinear then $CC(y,\bar{a})$ has degree at most 2 in each variable and total degree at most 4.

We cannot probe all 2^ℓ possible choices y. Instead for each assignment $\bar{a}$ we can think of the possible values of $CC(y,\bar{a})$ as a vector of length 2^ℓ which we want to be the vector 0^{2^ℓ}. The idea is to use a linear error-correcting code (that maps 0^{2^ℓ} to a zero vector) and any non-zero vector to a vector that has a constant fraction of non-zero entries so we will be able to detect whether or not the original vector was 0^{2^ℓ} with reasonable probability. The code we use here is a Reed-Muller code but many other codes would also work.

Choose ℓ random elements $R = (r_1, r_2, \ldots, r_\ell) \in_R I^\ell$ where $I \subseteq \mathbb{F}$ and define

$$E_R(a) = \sum_{y\in\{0,1\}^\ell} CC(y,a) \prod_{i \text{ s.t. } y_i=1} r_i.$$

For a fixed a and varying R, $E(a) = E(a, r_1, \ldots, r_\ell)$ is a multilinear polynomial in $r_1, r_2, \ldots, r_\ell$ with coefficients $CC(y,a)$. Moreover, $E(a)$ is the zero polynomial if and only if $\forall y \in \{0,1\}^\ell\ CC(y,a) = 0$. By the Schwartz-Zippel Lemma applied to $E(a)$, $\Pr_R[E_R(a) = 0] \le \frac{\ell}{|I|}$.

To check that φ is satisfiable, the verifier chooses a random R and checks that $E_R(\bar{a}) = 0$ using the following protocol.

Sum-Check interactive protocal of Lund-Fortnow-Karloff-Nisan: Given a multivariable polynomial p of max degree k verify $\sum_{y\in\{0,1\}^\ell} p(y) = c_0$ (where in our case, $c_0 = 0, k = 2$, and $p(y) = CC(y,\bar{a}) \prod_{i \text{ s.t. } y_i=1} r_i$).

Define $g_1(z) = \sum_{y_2,\ldots,y_\ell\in\{0,1\}} p(z, y_2, \ldots, y_\ell)$.

The prover sends the coefficients of a degree k polynomial f_1 claimed to be g_1. The verifier checks that $f_1(0) + f_1(1) = c_0$, chooses random $r'_1 \in_R I \subseteq \mathbb{F}$, sends r'_1 to prover, and sets $c_1 = f_1(r'_1)$.

At the next round, $g_2(z) = \sum_{y_3,\ldots,y_\ell\in\{0,1\}} p(r'_1, z, y_2, \ldots, y_\ell)$, the prover sends the coefficients of f_2, the check is that $f_2(0) + f_2(1) = c_1$ and so forth.

At the end, the verifier directly checks that the value of $p(r'_1, r'_2, \ldots, r'_\ell) = c_\ell$.

In the case of applying the sum-check protocol for checking that

$$E_R(\bar{a}) \ = \sum_{y\in\{0,1\}^\ell} CC(y,\bar{a}) \prod_{i \text{ s.t. } y_i=1} r_i \ = \ 0,$$

the values of $r_1, \ldots, r_\ell$ are known to the verifier. Once the values of $y_1, \ldots, y_\ell$ have been substituted by $r'_1, \ldots, r'_\ell$, the structure of $CC(y,a)$ ensures that $\bar{a}$ will only need to be queried in the three random places specified by $r'_1, \ldots, r'_{\ell-3}$. Thus the final check of the verifier can be done by the verifier with three queries to the purported multilinear extension $\bar{a}$ of a.

Proof Table: The above protocol is described interactively. However, the proof yields the following entries in the proof table. For each $r'_1, r'_2, \ldots, r'_{i-1} \in I$, for $1 \le i \le \ell$ the table contains coefficients of a degree $\le k$ polynomial, $g_{r'_1,\ldots,r'_{i-1}}(z) = \sum_{y_{i+1},\ldots,y_\ell} p(r'_1, \ldots, r'_{i-1}, z, y_{i+1}, \ldots, y_\ell)$.

The size of the proof table is $O(|I|^\ell \cdot k \cdot \log |I|)$ bits, where ℓ is $\Theta(\log n)$ and $|I|$ is $\log^{\Theta(1)}(n)$ and $k = 2$.

Overall, the table will have $|I|^\ell$ such sum-check proofs, one for each choice of $r'_1, \ldots, r'_\ell$.

There are a few details to fix up, such as counting queries and random bits, but as we have described the proof so far, the size of the table is still at least $|I|^{\Theta(\ell)} = \log n^{\Theta(\log n)} = n^{\Theta(\log\log n)}$ which is not polynomial.

We can modify the proof in the following way so that the space required is polynomial. Encode the variable names in base h; so that rather than using $\{0, 1\} \subseteq I \subseteq \mathbb{F}$, use $H \subseteq I \subseteq \mathbb{F}$ where $|H| = h = \log n$. In this way, one can reduce the number of field elements required to encode a variable or clause to $\ell' = O(\log n / \log\log n)$. This will have the advantage that $|I|^\ell$ will only be polynomial but it will have the drawback that instead of using multilinear extensions we will need to use extensions of maximum-degree $k = h - 1$. For example, it will mean that in the sum-check protocol the test will be that $\sum_{y \in H^\ell} p(y) = c_0$ and thus instead of checking that $f_i(0) + f_i(1) = c_{i-1}$ at each step, the verifier will need to check that $\sum_{j \in H} f_i(j) = c_{i-1}$.

The rest of the analysis including the total number of queries and random bits is sketched in the next lecture. □