11/5/2003

Notes for Lecture 6

6.1 Outline

Today's lecture will cover:

- Applications of List-Decoding of Hadamard Codes
  - Hard-core Predicates for One-way Permutations
  - Pseudorandom Generators from One-way Permutations
  - Learning Algorithm for Fourier Coefficients of a Function
- Hard-core predicates for RSA, EXP

6.2 Review of Goldreich-Levin Theorem

It is convenient to think of the codewords of the Hadamard code as functions:

Definition 1 *Let $a \in \{0,1\}^k$, and define $L_a : \{0,1\}^k \to \{0,1\}$ to be the function $L_a(x) = a \cdot x$. Then, $L_a(\cdot)$ is the Hadamard encoding of a.*

We may then state the list-decoding algorithm for the Hadamard code from the previous lecture as follows:

Theorem 1 *(Goldreich-Levin) There is a (probabilistic) algorithm that given oracle access to a function $g : \{0,1\}^k \to \{0,1\}$ and a parameter $\epsilon > 0$ runs in time $O\left(\frac{1}{\epsilon^4} k \log k\right)$ and outputs a list of $O\left(\frac{1}{\epsilon^2}\right)$ elements of $\{0,1\}^k$ such that: for every a for which L_a and g agree on $> \frac{1}{2} + \epsilon$ fraction of inputs, the probability that a is in the list is at least $3/4$.*

6.3 Hard-core Predicates for One-way Permutations

Intuitively, a function $f : \{0,1\}^k \to \{0,1\}^k$ is a one-way permutation if it is easy to compute but hard on average to invert. Note that f is a permutation on the set $\{0,1\}^k$, and not a function that permutes the bits of its input string. Formally,

Definition 2 *A permutation $f : \{0,1\}^k \to \{0,1\}^k$ is (s, ϵ)-one-way if*

- *there is an efficient algorithm that on input x outputs $f(x)$*
- *for all circuits $D : \{0,1\}^k \to \{0,1\}^k$ of size s, and for all sufficiently large k,*

$$\Pr_x[D(f(x)) = x] \le \epsilon$$

Often in applications such as the construction of pseudorandom generators in cryptography, we are interested in a hard-on-average decision problem. This motivates the notion of a hard-core predicate:

Definition 3 *A function $B : \{0,1\}^k \to \{0,1\}$ is a (s, ϵ) hard-core predicate for a permutation f if it is hard on average to compute $B(x)$ given $f(x)$, that is: for all circuits $D : \{0,1\}^k \to \{0,1\}$ of size s, and for all sufficiently large k,*

$$\Pr_x[D(f(x)) = B(x)] \leq 1/2 + \epsilon$$

6.3.1 Hard-core Predicates via Goldreich-Levin

The Goldreich-Levin theorem gives us a hard-core predicate for any one-way permutation (with a slight modification) whose hardness is closely related to that of the original permutation:

Theorem 2 *If f is a $(s, 1/s)$-one-way, then $B(x, r) = x \cdot r$ is a $(s^{\Omega(1)}, 1/s^{\Omega(1)})$ hard-core predicate for the permutation $f' : \{0,1\}^{2k} \to \{0,1\}^{2k}$, where $f'(x, r) = (f(x), r)$.*

Proof: The proof is by contradiction: given an algorithm A that computes the hard-core predicate B on a $1/2 + \epsilon$ fraction of its inputs, we produce an algorithm that inverts f on some $O(\epsilon)$ fraction of inputs, contradicting the one-way-ness of f for a suitable choice of parameters. The reduction is uniform, so we will present the proof for a uniform algorithm A; the result extends readily to circuits. More precisely, having a uniform reduction means that if we start with a one-way permutation that is hard to invert using polynomial-time algorithms (respectively polynomial-sized circuits), we obtain a hard-core predicate that is hard to predict using polynomial-time algorithms (respectively polynomial-sized circuits).

Now, suppose we are given an algorithm A that runs in time t (or a circuit of size t for the non-uniform setting) such that:

$$\Pr_{x,r}[A(f(x), r) = x \cdot r] \geq 1/2 + \epsilon$$

Then by an averaging argument, we have:

$$\Pr_x[\Pr_r[A(f(x), r) = x \cdot r] \geq 1/2 + \epsilon/2] \geq \epsilon/2$$

Fix any x such that

$$\Pr_r[A(f(x), r) = x \cdot r] \geq 1/2 + \epsilon/2$$

Note that as we vary r over $\{0,1\}^k$, $x \cdot r$ yields the Hadamard encoding of x and therefore $A(f(x), \cdot)$ yields a good approximation to this encoding. We can then recover a list of candidates for x via list-decoding. The precise reduction is as follows: On input $f(x)$,

1. Define $g(r) = A(f(x), r)$. For an $\epsilon/2$ fraction of the choices of x, $g(r)$ and $L_x(r)$ agree on $1/2 + \epsilon/2$ fraction over the choices of r.

2. Run the Goldreich-Levin algorithm using $g(\cdot)$ as an oracle, and with parameter $\epsilon/2$. This takes time $O\left(\frac{1}{\epsilon^4} k \log k\right)$ and computes a list of size $O\left(\frac{1}{\epsilon^2}\right)$.

3. For each element x' of the list, output x' if $f(x') = f(x)$.

For a $\epsilon/2$ fraction of the choices of x, this algorithm outputs x on input $f(x)$ with probability 3/4. This yields an algorithm A' that runs in time $O\left(t \cdot \frac{1}{\epsilon^4} k \log k + \frac{1}{\epsilon^2} k^{O(1)}\right)$ satisfying

$$\Pr_x[A(f(x)) = x] \geq 3\epsilon/8$$

which means f cannot be $(poly(k, 1/\epsilon, t), O(\epsilon))$-one-way. It follows that if f is $(s, 1/s)$-one-way, then B is a $(s^{\Omega(1)}, 1/s^{\Omega(1)})$ hard-core predicate for the permutation f'. $\square$

Remark 1 *Note that if f is one-way, then f' is also one-way.*

6.3.2 Hard-core Predicates via (Efficiently) List-Decodable Codes

Let $C : \{0,1\}^k \to \{0,1\}^n$ be a binary code, and $f : \{0,1\}^k \to \{0,1\}^k$ be a permutation. Consider the predicate $B : \{0,1\}^k \times [n] \to \{0,1\}$ given by $B(x, j) = C(x)_j$ and the corresponding function $f' : \{0,1\}^k \times [n] \to \{0,1\}^k \times [n]$ given by $f'(x, j) = (f(x), j)$. Extending the result in the previous section (which corresponds to C being the Hadamard code), it is easy to see that if f is a one-way permutation, and C is an efficiently list-decodable code, then $B(x, j) = C(x)_j$ is a hard-core predicate for f'. The reduction is as follows:

Suppose we are given an algorithm A that runs in time t such that:

$$\Pr_{x,j}[A(f(x), j) = C(x)_j] \geq 1/2 + \epsilon$$

Then,

$$\Pr_x[\Pr_j[A(f(x), j) = C(x)_j] \geq 1/2 + \epsilon/2] \geq \epsilon/2$$

Now, given $f(x)$, use the list-decoding for C to find a list of all codewords with agreement at least $1/2 + \epsilon/2$ to the corrupted codeword whose jth entry is $A(f(x), j)$ in time $poly(k, 1/\epsilon)$.

The advantage here is that for certain choices of ϵ, there are efficiently list-decodable whose block length is much shorter than that for the Hadamard code, and therefore we can construct hard-core predicates for one-way permutations f' whose input length is less than double that of f. For instance, we can concatenate Reed-Muller codes with Hadamard codes to obtain codes with message length k and block length $O(k^2/\epsilon^2)$ which are efficiently list-decodable from agreement $1/2 + \epsilon$, and for which we can compute $C(x)_j$ in time $poly(k, \log 1/\epsilon)$ (to ensure that the predicate can be computed efficiently). For $\epsilon = k^{-\log k}$, this yields a construction of hard-core predicates wherein the increase in the input length for f' is only $O(\log^2 k)$.

More generally, starting with a $(s, 1/s)$-one-way permutation $f : \{0,1\}^k \to \{0,1\}^k$, we can construct another $(s, 1/s)$-one-way permutation $f' : \{0,1\}^{k+O(\log s)} \to \{0,1\}^{k+O(\log s)}$ that has a $(s^{\Omega(1)}, 1/s^{\Omega(1)})$ hard-core predicate.

6.3.3 Pseudorandom Generators from One-way Permutations

It is not difficult to show that if $x \cdot r$ is hard to compute given $f(x), r$, then $(f(x), r, x \cdot r)$ is computationally indistinguishable from a random $(2k+1)$-bit string. In particular, if f

is a one-way permutation, then $G : \{0,1\}^{2k} \to \{0,1\}^{2k+1}$ that sends (x, r) to $(f(x), r, x \cdot r)$ is a pseudorandom generator.

In order to obtain a pseudorandom generator with any arbitrary polynomial stretch, say from $2k$ bits to $p(k)$ bits, we compose the generator G $p(k)$ times to obtain $G' : \{0,1\}^{2k} \to \{0,1\}^{p(k)}$, where the i-th bit of the output is $f^i(x) \cdot r$. It follows from a standard hybrid argument that G' is pseudorandom.

6.4 Learning Algorithm for Fourier Coefficients of a Function

6.4.1 Review of Fourier Analysis

In the context of list-decoding of Hadamard codes, we have been looking at boolean functions $f : \{0,1\}^k \to \{0,1\}$, and linear functions $L_a : \{0,1\}^k \to \{0,1\}$, where $L_a(x) = \oplus_{i:a_i=1} x_i$ Instead, we may regard f as a function $f : \{0,1\}^k \to \{1,-1\} \subseteq \mathbb{R}$ by identifying $0,1$ with $1,-1$ in the range, and $\oplus$ with $\cdot$ (multiplication over $\mathbb{R}$). In addition, consider $\chi_a : \{0,1\}^k \to \{1,-1\}$ where $\chi_a(x) = \prod_{i:a_i=1}(-1)^{x_i} = (-1)^{x \cdot a}$.

For any $f,\ g : \{0,1\}^k \to \mathbb{R}$, we define the dot product $f \cdot g$ to be given by $\frac{1}{2^k}\sum_x f(x)g(x)$. It is then straight-forward to verify that $\chi_a \cdot \chi_a = 1$, and $\chi_a \cdot \chi_b = 0$ for $a \neq b$. Therefore, $\{\chi_a \mid a \in \{0,1\}^k\}$ is an orthonormal basis for the set of functions $f : \{0,1\}^k \to \mathbb{R}$. This means we can write any function $f : \{0,1\}^k \to \mathbb{R}$ as:

$$f(x) = \sum_{a \in \{0,1\}^k} \hat{f}_a \chi_a(x)$$

where $\hat{f}_a = f \cdot \chi_a \in \mathbb{R}$.

The dot product $f \cdot g$ and the Fourier coefficients $\hat{f}_a$ have a combinatorial interpretation when $f, g : \{0,1\}^k \to \{1,-1\}$ are boolean functions. Observe that

$$f \cdot g = \frac{1}{2^k}\sum_x f(x)g(x) = \Pr_x[f(x) = g(x)] - \Pr_x[f(x) \neq g(x)] = 2\Pr_x[f(x) = g(x)] - 1$$

Therefore,

$$\Pr_x[f(x) = g(x)] = \frac{1}{2} + \frac{1}{2} f \cdot g$$

and in particular,

$$\Pr_x[f(x) = L_a(x)] = \frac{1}{2} + \frac{1}{2}\hat{f}_a \tag{1}$$

Also, for any $f : \{0,1\}^k \to \mathbb{R}$, we have $f \cdot f = \sum_a \hat{f}_a^2$. In the case f is a boolean function, $f \cdot f = 1$, so this yields:

$$\sum_a \hat{f}_a^2 = 1 \tag{2}$$

6.4.2 Learning Boolean Functions

The Goldreich-Levin theorem may be interpreted as providing an efficient algorithm that given oracle access to some boolean function f and some threshold ϵ, finds all linear functions L_a that agrees with f on at least $1/2 + \epsilon$ fraction of inputs. By (1), these linear functions correspond exactly to Fourier coefficients $\hat{f}_a$ that are at least 2ϵ. Furthermore, given a and oracle access to f, estimating $\hat{f}_a$ is easy (by estimating $\Pr_x[f(x) = L_a(x)]$), so we may eliminate any extraneous coefficients that may be computed by the Goldreich-Levin algorithm.

This idea is often applied to learning classes of boolean functions f for which the Fourier coefficients are concentrated on a small set S, that is, there is some set $S \subseteq \{0,1\}^k$ such that $|S| = poly(k)$ and $\sum_{a\in S} \hat{f}_a^2 \geq 1 - \epsilon$. Given oracle access to access to f, we can define a function g such that f and g disagree on $O(\epsilon)$ fraction of inputs. Furthermore, g can be efficiently computed as follows:

1. Fix $t = poly(k, 1/\epsilon)$.
2. Find a list L of all a such that the corresponding Fourier coefficients $\hat{f}_a$ are at least ϵ/t using the Goldreich-Levin algorithm. From (2), there are at most $O(t^2/\epsilon^2)$ such values.
3. Compute an estimate $\hat{f}'_a$ of $\hat{f}_a$ for all $a \in L$ by estimating $\Pr_x[f(x) = L_a(x)]$ using sampling.
4. Return $g = \sum_{a\in L} \hat{f}'_a \chi_a$ as an estimate for f.

To bound the fraction of inputs on which f and g disagree, we will need to bound the errors due to the omission of the small Fourier coefficients, and in the estimation of the large Fourier coefficients.

6.5 Fourier Analysis for $\mathbb{Z}_N$

We will begin by introducing an analogue to Fourier analysis for $\mathbb{Z}_N$ and complete the general construction of hard-core predicates for RSA, EXP in the next lecture, which is based on a learning algorithms for the Fourier coefficients over $\mathbb{Z}_N$, where N may either be prime or composite.

Consider the set of functions $f : \mathbb{Z}_N \to \mathbb{C}$, which forms an N-dimensional space. Let ω be a primitive N-th root of unity, say $\omega = e^{2\pi i/N}$, so that $\omega, \omega^2, \ldots, \omega^N = 1$ are distinct. For any $a \in \mathbb{Z}_N$, define $\chi_a : \mathbb{Z}_N \to \mathbb{C}$ given by $\chi_a(x) = \omega^{ax}$.

For any $f, g : Z_N \to \mathbb{C}$, we define the dot product $f \cdot g$ to be given by $\frac{1}{N}\sum_x f(x)\overline{g(x)}$. It is then straight-forward to verify that $\chi_a \cdot \chi_a = 1$, and $\chi_a \cdot \chi_b = 0$ for $a \neq b$. Therefore, $\{\chi_a \mid a \in \mathbb{Z}_N\}$ is an orthonormal basis for the set of functions $f : \mathbb{Z}_N \to \mathbb{C}$. This means we can write any function $f : \mathbb{Z}_N \to \mathbb{C}$ as:

$$f(x) = \sum_{a\in\mathbb{Z}_N} \hat{f}_a \chi_a(x)$$

where $\hat{f}_a = f \cdot \chi_a \in \mathbb{C}$.