PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

# Lecture 10-11: One-way permutations and Goldreich-Levin bit theorem

Lecturer: *Sanjeev Arora* Scribe:*Edith Elkind*

-11

## 1 Proof of Goldreich–Levin theorem (continued)

In the last lecture, we formulated Goldreich–Levin theorem, which says that if $\{f_n\}$ is a one-way permutation, then the mapping $(x, r) \mapsto (f(x), r, x \odot r)$ extends $2n$ bits to $2n+1$ bits in a pseudorandom fashion. More formally, for all algorithms $A$ running in time $s^{1/4}(n)$

$$\mathbf{Pr}_{x,r\in\{0,1\}^n}[A(f_n(x), r) = x \odot r] \leq \frac{1}{2} + O\left(\frac{1}{s(n)}\right). \tag{1}$$

PROOF:[continued from Lecture 9] Last time, we gave a proof for the case when RHS of (1) is $3/4 + \delta$. The idea for the general case is very similar, the only difference being that this time we want to pick $r_1, \ldots, r_m$ so that we already "know" $x \odot r_i$. The preceding statement may appear ridiculous, since knowing the inner product of $x$ with $m \geq n$ random vectors is, with high probability, enough to reconstruct $x$ (check this!). The catch will of course be that the $r_i$'s will not be completely random. Instead, they will be pairwise independent.

DEFINITION 1 *Random variables $x_1, \ldots, x_m$ are* pairwise independent *if*

$$\forall i, j\ \forall a, b\ \mathbf{Pr}[x_i = a, x_j = b] = \mathbf{Pr}[x_i = a]\mathbf{Pr}[x_j = b]. \tag{2}$$

Pairwise independent random variables are useful because of the Chebyshev inequality. Suppose that $x_1, \ldots, x_m$ are pairwise independent with $E(x_i) = \mu$, $Var(x_i) = \sigma^2$. Then we have

$$Var(\sum x_i) = E[(\sum_i x_i)^2] - E[\sum_i x_i]^2 = \sum_i (E[x_i^2] - E[x_i]^2),$$

where we have used pairwise independence in the last step. Thus $Var(\sum x_i) = m\sigma^2$. By the Chebyshev inequality, $\mathbf{Pr}[|\sum x_i - m\mu| > k\sqrt{m}\sigma] \leq 1/k^2$. So, the sum of the variables is somewhat concentrated about the mean. This is in contrast with the case of complete independence, when Chernoff bounds would give an exponentially stronger concentration result (the $1/k^2$ would be replaced by $\exp(\Theta(-k^2))$).

EXAMPLE 1 In $\mathbf{Z}_p$, choose $a$ and $b$ randomly and independently. Then the random variables $a+b, a+2b, \ldots, a+(p-1)b$ are pairwise independent. Indeed, for any $t, s, j \neq k \in \mathbf{Z}_p$, $\mathbf{Pr}[a + jb = t] = 1/p$, and $\mathbf{Pr}[a + jb = t, a + kb = s] = 1/p^2$, because this linear system is satisfied by exactly one $(a, b)$-pair out of $p^2$.

EXAMPLE 2 Let $m = 2^k - 1$. The set $[1 \ldots m]$ is in $1-1$ correspondence with the set $2^{[1\ldots k]} \backslash \emptyset$. We will construct $m$ random variables corresponding to all nonempty subsets of $[1 \ldots k]$.

Pick uniformly at random $k$ binary strings $t_1, \dots, t_k$ of length $n$ and set $Y_S = \sum_{i \in S} t_i \pmod 2$, where $S \subset [1 \dots k]$, $S \neq \emptyset$. For any $S_1 \neq S_2$, the random variables $Y_{S_1}$ and $Y_{S_2}$ are independent, because one can always find an $i$ such that $i \in S_2 \setminus S_1$, so the difference between $Y_{S_1}$ and $Y_{S_2}$ is always a sum of several uniformly distributed random vectors, which is a uniformly distributed random vector itself. That is, even if we fix the value of $Y_{S_1}$, we still have to toss a coin for each position in $Y_{S_2}$, and

$$\mathbf{Pr}[Y_{S_1} = \vec{s}, Y_{S_2} = \vec{t}] = \mathbf{Pr}[Y_{S_1} \oplus Y_{S_2} = \vec{t} \oplus \vec{s} | Y_{S_1} = \vec{s}] = \frac{1}{2^{2n}}.$$

Now let us return to the proof of Goldreich–Levin theorem and describe the observation at the heart of the proof. Suppose that our random strings $r_1, \dots, r_m$ are $\{Y_S\}$ from the previous example. Then $x \odot Y_S = x \odot (\sum_{i \in S} t_i) = \sum_{i \in S} x \odot t_i$. Now, if we know $x \odot t_i$ for $i = 1, \dots, k$, we also know $x \odot Y_S$. Of course, we don't know $x \odot t_i$ for $i = 1, \dots, k$, but we can just try all $2^k$ possibilities for this vector and run the rest of the algorithm for each of them. This multiplies the running time by a factor $2^k$, which is only $m$. This is how we can assume that we know $x \odot Y_S$ for each subset $S$.

The details of the rest of the algorithm are similar to before. By Lemma 2, we know that if the theorem were not true, then for at least $\delta = \alpha/s(n)$ fraction of $x$'s, where $\alpha$ is some constant, the probability over $r$ that $A$ gives the correct answer is at lest $1/2 + \delta/2$. We now concentrate our attention on those $x$'s. Pick $m$ pairwise independent vectors $Y_S$'s as described above, calculate $A(f_n(x), Y_S \oplus e_i) - x \odot Y_S$, and take the majority vote. The expected number of correct answers is $m(1/2+\delta/2)$, so for the majority vote to result in the incorrect answer it must be the case that the number of incorrect values deviates from its expectation by more than $m\delta/2$. Now, we can bound the variance of this random variable and apply Chebyshev's inequality.

Formally, let $\xi_S$ denote the event that $A$ produces the correct answer on $Y_S$; we have $E(\xi_S) = 1/2+\delta/2$ and $Var(\xi_S) = E(\xi_S)(1-E(\xi_S)) < 1$. Let $\xi = \sum_S \xi_S$ denote the number of correct answers on a sample of size $m$. By linearity of expectation, $E[\xi] = m(1/2+\delta/2)$. Furthermore, the $Y_S$'s are pairwise independent, which implies that the same is true for the outputs $\xi_S$'s produced by the algorithm $A$ on them. Hence by pairwise independence $Var(\xi) < m$. Now, by Chebyshev's inequality, the probability that the majority vote is incorrect is at most $\frac{4Var(\xi)}{m^2\delta^2} \leq \frac{4}{m\delta^2}$. Recalling that $\delta = \alpha/s(n)$, we see that if we set $m = \Omega(ns^2(n))$, the probability of guessing the $i$th bit incorrectly is at most $1/2n$, and by the union bound, the probability of guessing the whole word incorrectly is at most $1/2$. Hence, on a "good" $x$, we can find the preimage of $f(x)$ with a good probability, and the number of "good" $x$'s is non-negligible, which contradicts our assumption that $f$ is one-way. □

## 2 Applications

### 2.1 Playing poker over the phone

How can two parties $A$ and $B$ play poker over the phone? Specifically, how do they deal the cards in a fair way? Clearly, they need a way to toss a fair coin over the phone. If only one of them actually tosses a coin, there is nothing to prevent him from lying about the result. The following fix suggests itself: both players toss a coin and they take the XOR as the shared coin. Even if $B$ does not trust $A$ to use a fair coin, he knows that as long as his bit is random, the XOR is also random. Unfortunately, this idea does not work because the player who reveals his bit first is at a disadvantage: the other player could just "adjust" his answer to get his desired coin toss.

This problem is addressed by the following scheme, which assumes that $A$ and $B$ are polynomial time turing machines that cannot invert one-way permutations. The protocol itself is called *bit commitment.* First, $A$ chooses two strings $x_A$ and $r_A$ of length $n$ and sends a message $(f_n(x_A), r_A)$, where $f_n$ is a one-way permutation. This way, $A$ commits the string $x_A$ without revealing it. Similarily, $B$ chooses $x_B$ and $r_B$ and sends $A$ a message $(f_n(x_B), r_B)$. After that, both parties are ready to reveal their strings, so $A$ sends $B$ a message $(x_A, x_A \odot r_A)$, and $B$ sends $A$ a message $(x_B, x_B \odot r_B)$. Here, $x_A \odot r_A$ serves as $A$'s coin toss; $B$ can verify that $x_A$ is the same as in the first message by applying $f_n$, therefore $A$ cannot change her mind after learning $B$'s bit. On the other hand, by Goldreich–Levin theorem, $B$ cannot predict $x_A \odot r_A$ from $A$'s first message, so this scheme is secure.

Note that the second stage of this protocol is redundant: $B$ can simply announce his bit after receiving $A$'s first message.

## 2.2 Pseudorandom generation

Now we describe another application of one-way functions: to "stretch" $n$ truly random bits to obtain $n^c$ random-looking bits?

First, we have to define what it means for a string to look random. Kolmogorov gave one definition ("the length of the smallest Turing machine that outputs this string when started on an empty tape") but that is not very useful because of noncomputability issues. Blum and Micali proposed instead that we should define randomness for *distributions* rather than for strings. The distribution is declared pseudorandom if its samples "look" random to every polynomial time Turing machine. The next definition (due in this form to Yao) formalizes this notion.

Definition 2 (Yao'82) *A family $\{g_n\}$, $g_n$ $\{0,1\}^n \mapsto \{0,1\}^{n^c}$, is called a* pseudorandom generator *if for any algorithm $A$ running in time $s(n)$ and for all large enough $n$*

$$|\mathbf{Pr}_{y\in\{0,1\}^{n^c}}[A(y) = \textit{accept}] - \mathbf{Pr}_{x\in\{0,1\}^n}[A(g_n(x)) = \textit{accept}]| \le \delta(n), \qquad (3)$$

*where $\delta(n)$ is the* distinguishing probability *and $s(n)$ is the* security parameter*.*

Suppose that $f_n$ is a one-way permutation with security $s(n)$, and distinguishing probability $1/s(n)$, say. Then a pseudorandom generator can be built as follows. Take $2n$ random bits; denote the first $n$ bits by $x$ and the last $n$ bits by $r$. Construct a string $(f_n(x), r, x \odot r)$; repeat the same procedure with the last $2n$ bits of this string; keep doing this until you get $n^c$ bits.

Theorem 1
*No algorithm running in time $\frac{s(n)^{1/4}}{n^c}$ can distinguish a string obtained in this way from a random string with probability higher than $\frac{2n^c}{s(n)}$.*

Proof:[Yao's hybrid argument] By $D_0$ denote the distribution $\{g(z)\}$, $z \in \{0,1\}^{2n}$, and by $F$ denote the uniform distribution on $n^c$ bits. We are going to construct a sequence of distributions that starts with $D_0$ and gradually transforms it to $F$. Namely, let $D_i$, $i = 1, \ldots, n^c$ be a distribution in which the first $i$ bits are random, while other bits are obtained by applying $g$ to the $2n$ bits that precede them, just as we did for $D_0$ (or, if $i \leq 2n$, the bits from $i+1$st to $2n$th position are the last $2n - i$ bits of $f_n(z)$). Note that $D_{n^c} = F$. Also, since $f_n$ is a permutation, the first $2n$ bits of $D_0$ are uniformly distributed, so $D_1 = \ldots = D_{2n} = D_0$.

Now, suppose that there is an algorithm that can distinguish between $D_0$ and $D_{n^c}$. Then this algorithm can also distinguish between $D_i$ and $D_{i+1}$ for some $i$, which means that it can predict Goldreich–Levin bit. More formally, suppose that there is an algorithm $A$ such that

$$|\mathbf{Pr}_{y \in D_{n^c}}[A(y) = \text{accept}] - \mathbf{Pr}_{y \in D_0}[A(y) = \text{accept}]| \geq \delta. \tag{4}$$

Then there exists an $i$ such that

$$|\mathbf{Pr}_{y \in D_i}[A(y) = \text{accept}] - \mathbf{Pr}_{y \in D_{i+1}}[A(y) = \text{accept}]| \geq \frac{\delta}{n^c}. \tag{5}$$

The only difference between $D_i$ and $D_{i+1}$ is in the $i$th bit: in $D_i$, it is obtained from $2n$ previous bits by Goldreich–Levin construction, while in $D_{i+1}$ it is random.

Consider an algorithm $B$ that given $f(x)$, $r$, and a bit $b_0$ (allegedly, $x \odot r$) constructs a string that starts with $i - 2n$ random bits followed by $f(x)$, $r$, and $b_0$; the remaining bits are produced as described above. Obviously, for random $x$ and $r$ if, indeed, $b_0 = x \odot r$, this string is distributed according to $D_i$, while if $b_0$ is uniformly distributed over $\{0,1\}$, this string is distributed according to $D_{i+1}$. Then $B$ runs $A$ on this string and accepts iff $A$ accepts.

We have $\mathbf{Pr}_{x,r,r'}[B(f(x), r, x \odot r) = \text{accept}] = p_1$, $\mathbf{Pr}_{x,r,b,r'}[B(f(x), r, b) = \text{accept}] = p_2$, where $r'$ stands for $B$'s internal coin tosses. Without loss of generality, $p_2 < p_1$, hence $p_2 < p_1 - \delta/n^c$. Note that $\mathbf{Pr}_{x,r,b}[B(f(x), r, b) = \text{accept}]= \mathbf{Pr}_{x,r}[B(f(x), r, x \odot r) = \text{accept}] \times \frac{1}{2}+$ $\mathbf{Pr}_{x,r}[B(f(x), r, \overline{x \odot r}) = \text{accept}] \times \frac{1}{2}$, nence if $p_2 < p_1 - \delta$, then $\mathbf{Pr}_{x,r}[B(f(x), r, \overline{x \odot r}) = \text{accept}] \leq p_1 - 2\delta$. An averaging argument similar to the one given in the previous lecture shows that for a non-negligible fraction of $x$'s we can find the Goldreich-Levin bit with a significant probability by the following procedure. Run $B$ several times and accept if $B$ accepts on at least $p_1 - \delta$ fraction of the input. The correctess of this approach can be justified by Chernoff or Chebyshev inequality. □

As a corollary, Yao could give another definition of pseudorandom generators: they are distributions that pass the "next bit test" (i.e., that given the first $i$ bits, a polynomial time algorithm cannot predict the $i+1$th bit with good probability).

Corollary 2
*A pseudorandom generator is secure if and only if it passes the next bit test.*

Proof: A hybrid argument again, only this time the random bits are shifted in from right to left. □

Theorem 3
*If there is a pseudorandom generator that is secure against circuits of size $n^c$, then* $\mathbf{BPP} \subseteq \cap_{\varepsilon>0}\mathbf{DTIME}(2^{n^\varepsilon})$.

In words, pseudorandom generators imply subexponential algorithms for **BPP**. For this reason, this theorem is usually referred to as *derandomization* of **BPP**.
PROOF: Let us fix an $\varepsilon > 0$ and show that $\mathbf{BPP} \subseteq \mathbf{DTIME}(2^{n^\varepsilon})$.

Suppose that $M$ is a **BPP** machine running in $n^k$ time. We can build another probabilistic machine $M'$ that takes $n^\varepsilon$ random bits, streches them to $n^k$ bits using the pseudorandom generator and then simulates $M$ using this $n^k$ bits as a random string. Obviously, $M'$ can be simulated by going over all binary strings $n^\varepsilon$, running $M'$ on each of them, and taking the majority vote.

It remains to prove that $M$ and $M'$ accept the same language. Suppose otherwise. Then there exists an infinite sequence of inputs $x_1, \ldots, x_n, \ldots$ on which $M$ distinguishes a truly random string from a pseudorandom string with a high probability, because for $M$ and $M'$ to produce different results, the probability of acceptance should drop from 2/3 to below 1/2. Hence we can build a distinguisher similar to the one described in the previous theorem by hardwiring these inputs into a circuit family. $\Box$

The relationship between hardness and randomness is a subject of recent research.

## 3 Average-case hardness

One-way functions and pseudorandom generators were covered in this course to give some idea of complexity theory as applied to the "average" case. Leonid Levin has developed a more general theory of average-case complexity. The main difference from the above study is that he defines a distribution over all (infinitely many) input strings rather than just a distribution over strings of a fixed length as we have done here (and which suffices for cryptography since typically the key-size used in a publicly available package such as DES is fixed).

Levin (1986) defines a distributional problem as a pair $(\pi, \mathcal{D})$, where $\pi$ is a decision problem and $\mathcal{D}$ is a polynomial-time samplable distribution on inputs[1]. Levin's class *Avg-P* contains $\pi$ for which $(\pi, \mathcal{D})$ has a polynomial-time average case algorithm for every $\mathcal{D}$. There are some subtleties in this definition that we will not explore.

Interestingly, Levin can prove that certain distributional problems are complete for this class under probabilistic polynomial time reductions. Many of these problems are specialized and not very natural. A very interesting and largely open problem is whether TSP or any other natural hard problem is complete (or hard) for Levin's class.

[1]The restriction to polynomial-time samplable distributions seems reasonable if we believe in the strong form of the Church-Turing thesis, which asserts that the probabilitic Turing machine can simulate every physically realizable computational model with polynomial slowdown. Then we can view the world as a simulatable system, and the instances produced by it it would then come from a polynomial-time samplable distribution.