12 Pseudo-Random Number Generators (April 15)

Communication complexity with bias has the following remarkable application, first developed by Babai, Nisan, and Szegedy. Random numbers are used throughout theoretical computer science as a way to make algorithms simpler and (possibly) more efficient, but in the real world, it is extremely difficult to generate truly random numbers. In practice, people use so-called pseudo-random number generators: simple formulas for transforming an initial truly random seed value into a sequence of bits that hopefully appears random enough to have the same behavior as truly randomness. In fact, most pseudo-random number generators are horrible; the standard C library function `rand()` produces numbers that are alternately even and odd! Our goal is to describe a pseudo-random number generator with the property that no simple predictor is correct more than 50.0001% of the time.

12.1 Definitions and the Main Theorem

More formally, a generator is a function of the form

$$g : \{0, 1\}^m \to \{0, 1\}^n$$

where $n \gg m$. Note that there is no randomness in this definition; if the input string $x$ is uniformly distributed over $\{0, 1\}^m$, however, the generator induces a probability distribution over $\{0, 1\}^n$ that is concentrated on at most $2^m$ different output strings $g(x)$.

A predictor is a function of the form

$$P : \bigcup_{i=1}^{n} \{0, 1\}^{i-1} \to \{0, 1\}^i$$

that predicts, given a proper prefix of a generator’s output, the next generated bit. We say that a predictor $P$ is $\varepsilon$-good against a generator $g$, for some $0 < \varepsilon \leq 1/2$, if the probability that $P$ is correct is at least $1/2 + \varepsilon$, when the input to $g$ is chosen uniformly at random.

We need to fix our model of computation. An online Turing machine has a one-way, read-only input tape and a two-way, read-write work tape. The input head starts at the leftmost input bit and always moves to the right. The space used by an online Turing machine is the number of cells used on the work tape. From now on, we assume that any predictor is an online Turing machine.

**Theorem 1 (Babai, Nisan, and Szegedy).** For any $0 < \varepsilon \leq 1/2$, there is a simple generator $g$ such that any $\varepsilon$-good predictor for $g$ uses space $2^{\Omega(\sqrt{\log n})}$.

Theorem 1 replies on the following result about multiparty communication protocols with bias, which I won’t prove. The generalized inner product function $G^r_k$ is defined as

$$G^r_k(x_1, \ldots, x_k) = \left(\sum_{j=1}^{r} \prod_{i=1}^{k} x_{ij}\right) \mod 2,$$

where $x_{ij}$ is the $j$th bit of the $i$th input string. In other words, the generalized inner product is the parity of the bitwise AND of the strings. The communication complexity of $G^r_k$ with bias $\varepsilon$, denoted $C^r_\varepsilon(G^r_k)$, is the minimum number of bits in any protocol that computes $G^r_k$ for at least $1/2 + \varepsilon$ of the possible inputs. (Observe that we’ve significantly changed the role of $\varepsilon$ here!)
Lemma 2 (Babai, Nisan, and Szegedy). \( C_\varepsilon(G_k^n) = \Omega(r/c^k) \) for some constant \( c < 8 \), where the hidden constant depends on \( \varepsilon \).

Before we get to the actual proof, let’s describe the “simple” generator \( g = g(r, k, t) \) in terms of three parameters \( r \), \( k \), and \( t \) to be determined later. Our generator \( g \) has the form

\[
g : (\{0,1\}^r)^t \rightarrow \{0,1\}^n;
\]

the input consists of \( t \) strings \( x_1, x_2, \ldots, x_t \), each of length \( r \). Let \( L \) be a list of all \( \binom{t}{k} \) subsets of \([t]\) of size \( k \) in reverse lexicographic order. For example, when \( t = 5 \) and \( k = 3 \), we have

\[
L = \langle 123, 124, 134, 234, 125, 135, 235, 145, 245, 345 \rangle
\]

To determine its \( i \)th output bit, \( g \) looks at the \( i \)th entry in list \( L \), which is a set of indices \( \{i_1, \ldots, i_k\} \); the \( i \)th output bit is the generalized inner product \( G_k(x_{i_1}, \ldots, x_{i_k}) \) of the corresponding input strings. We assume here that \( n < \binom{t}{k} \).

12.2 Reduction to Multi-Party Communciation

The following lemma forms the core of the proof of Theorem 1.

Lemma 3. Suppose there is a predictor \( P \) that is \( \varepsilon \)-good against \( g(r, k, t) \) and uses space \( s \). Then there is a \( k \)-party communication protocol for \( G_k^n \) with bias \( \varepsilon \) of complexity \( O(ks) \).

Proof: Let \( X = \{x_1, x_2, \ldots, x_k\} \) be a set of \( k \) random \( r \)-bit strings, which we will use as the input to our \( k \)-party protocol. To use the generator \( g(r, k, t) \), we need a seed consisting of \( t \) \( r \)-bit strings. To obtain this, we will choose in advance a set \( Y = \{y_1, \ldots, y_{k-1}\} \) of fixed \( r \)-bit strings; the seed will be \( X \cup Y \).

By definition the protocol \( P \) has probability \( 1/2 + \varepsilon \) of correctly predicting the output of \( g \) when the seed is chosen uniformly at random, but now we are using a different probability distribution, where only \( kr \) bits are random and the other \( (t-k)r \) bits are fixed. Let \( P(Y) \) be the probability of success for a given set \( Y \) and a uniformly random set \( X \). The average of \( P(Y) \) over all sets \( Y \) is the same as the success probability when the entire seed is chosen uniformly at random, that is, at least \( 1/2 + \varepsilon \). But then we must have \( P(Y) \geq 1/2 + \varepsilon \) for some particular set \( Y \). In other words, there is a fixed set \( Y \) such that \( P \) has probability \( 1/2 + \varepsilon \) of correctly predicting the output of \( g \) when only \( X \) is chosen uniformly at random.

To simplify the notation, we define \( y_{t-i+1} = x_i \) for all \( i \), so that the component strings in the seed can be indexed \( y_1, y_2, \ldots, y_t \). With this indexing, \( X \) is the \( k \)-element set chosen by the very last element of the list \( L \).

We can partition the list \( L \) into \( k + 1 \) different ranges \( L_1, L_2, \ldots, L_{k+1} \), so that for any \( i \leq k \), none of the subsets in range \( L_i \) contains the integer \( t - i + 1 \), and \( L_k = \{t - k + 1, \ldots, t\} \). Range \( L_1 \) will be quite large in general, and later ranges \( L_i \) will get smaller as \( i \) increases. For example, with \( t = 5 \) and \( k = 3 \), we have the following partition:

\[
L_1 = \langle 123, 124, 134, 234 \rangle \quad \text{[no 5]}
\]
\[
L_2 = \langle 125, 135, 235 \rangle \quad \text{[no 4]}
\]
\[
L_3 = \langle 145, 245 \rangle \quad \text{[no 3]}
\]
\[
L_4 = \langle 345 \rangle
\]
Finally, we divide the execution of the generator $g$ into phases, where each phase uses a different range of $L$.

Now back to our $k$-party protocol for $G_k^r$. Every player knows the strings $y_1, \ldots, y_{t-k}$ in addition to the other players’ input strings. In particular, the first player knows all the strings $y_1, \ldots, y_{t-1}$, but not $y_t = x_1$. Thus, the first player has enough information to evaluate the generator function for its entire first phase. Similarly, the $i$th player has enough information to run the generator during phase $i$.

The protocol works as follows. Player 1 computes the output bits of the generator during phase 1, then simulates the predictor generated bit in phase 2, he transmits the configuration of the next player, so it does not need to be transmitted.) If player until player $i$, but not player $i + 1$.

Let $k < 8$ be the constant in Lemma 2, and choose a constant $1 < a < 8/c$ such that $r/c^k \geq k a^k$. Then we have

$$r/c^k \geq k a^k = k \cdot 2^{k \lg n} = 2^{k/2} = 2^{\Omega(\log n)};$$

so by Lemma 2, any multiparty protocol for $G_k^r$ has complexity at least $k \cdot 2^{\Omega(\log n)}$. By Lemma 3, if there good predictor that uses space $s$, then there is a multiparty protocol for $G_k^r$ with length $O(k s)$, so we must have $s = 2^{\Omega(\log n)}$, as claimed. □
With these parameters, we have a generator that takes a seed of $16^{2\sqrt{\lg n}}$ random bits as input and produces $n$ pseudo-random bits as output. If $n < 2^{64}$, then the seed is actually longer than the output! To quote Dick Karp, this generator is practical only in the faraway fantasy land of Asymptopia.

However, by modifying the parameters, we can obtain a much more practical generator that still defeats any small-space generator. Following the proof carefully, we find that any $\varepsilon$-good predictor for $g(r, k, t)$ must use space $\Omega(rc^{-k}/k)$. So now all we have to do is find functions $r, t, k$ of $n$ so that $\binom{t}{k} > n > rt$ for all $n$ larger than some small integer $n_0$, but where $s = \Omega(kc^{-k}/k)$ is still big. Hmm....