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 ε*-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 ε-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_k^r is defeined as

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

where x_{ij} is the jth bit of the ith input string. In other words, the generalized inner product is the parity of the bitwise AND of the strings. The communication complexity of G_k^r with bias ε, denoted $C_\varepsilon(G_k^r)$, is the minimum number of bits in any protocol that computes G_k^r for at least $1/2 + \varepsilon$ of the possible inputs. (Observe that we've significantly changed the role of ε here!)

Lemma 2 (Babai, Nisan, and Szegedy). *$C_\varepsilon(G_k^r) = \Omega(r/c^k)$ for some constant $c < 8$, where the hidden constant depends on ε.*

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

$$g : (\{0,1\}^r)^t \to \{0,1\}^n;$$

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 ith output bit, g looks at the ith entry in list L, which is a set of indices $\{i_1, \ldots, i_k\}$; the ith output bit is the generalized inner product $G_k^r(x_{i_1}, \ldots, x_{i_k})$ 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 Thereom 1.

Lemma 3. *Suppose there is a predictor P that is ε-good against $g(r, k, t)$ and uses space s. Then there is a k-party communciation protocol for G_k^r with bias ε 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 uses 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 $Y = \{y_1, \ldots, y_{k-t}\}$ of *fixed* r-bit strings; the seed will be $X \cup Y$.

By definition the protocol P has probabilty $1/2+\varepsilon$ of corrrectly 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 probabilty $1/2 + \varepsilon$ of corrrectly 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:

$$\begin{aligned}
L_1 &= \langle 123, 124, 134, 234\rangle && \text{[no 5]}\\
L_2 &= \langle 125, 135, 235\rangle && \text{[no 4]}\\
L_3 &= \langle 145, 245\rangle && \text{[no 3]}\\
L_4 &= \langle 345\rangle
\end{aligned}$$

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 ith 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 P during that phase. When the P attempts to read the first generated bit in phase 2, he transmits the configuration of P to player 2. Player 2 computes the bits generated by g in phase 2, and continues the simulation. This process continues from player to player until player k computes P's prediction for the very last generated bit. The last bit preduced by the generator is actually $G_k^r(x_1, \ldots, x_k)$, and since P has bias ε, player k broadcasts this bit with probability at least $1/2 + \varepsilon$.

$$\begin{array}{ll}
\text{Player 1 computes} \left\{ \begin{array}{l} b_1 = G_3^r(y_1, y_2, x_3) \\ b_2 = G_3^r(y_1, y_2, x_2) \\ b_3 = G_3^r(y_1, x_3, x_2) \\ b_4 = G_3^r(y_2, x_3, x_2) \end{array} \right. \\
\text{Player 2 computes} \left\{ \begin{array}{l} b_5 = G_3^r(y_1, y_2, x_1) \\ b_6 = G_3^r(y_1, x_3, x_1) \\ b_7 = G_3^r(y_2, x_3, x_1) \end{array} \right. \\
\text{Player 3 computes} \left\{ \begin{array}{l} b_8 = G_3^r(y_1, x_2, x_1) \\ b_9 = G_3^r(y_2, x_2, x_1) \end{array} \right. \\
\text{Player 3 predicts} \quad b_{10} = G_3^r(x_3, x_2, x_1)
\end{array}$$

How many bits were transmitted by this protocol? At the end of each of the first $k-1$ phases, the entire configuration of the Turing machine P is broadcast: the state, the positions of the work tape head, and the contents of the work tape. (The position of the input tape head is known to the next player, so it does not need to be transmitted.) If P uses s cells of its work tape, we can encode the configuration in at most $O(1) + \lg s + s$ bits. Thus, the total number of bits transmitted is at most $ks + k \lg s + O(k) = O(ks)$. □

Now we can prove the main therorem.

Proof of Theorem 1 (from Lemma 3): Let n be any positive integer, and set

$$k = 2\sqrt{\lg n} \qquad t = 2^k \qquad r = 8^k$$

Note that in accordance with our earlier assumption, we have

$$\binom{t}{k} = \binom{2^k}{k} \approx 2^{k^2} = 2^{4 \lg n} = n^4 \gg n.$$

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

$$r/c^k \geq ka^k = k2^{k \lg a} = k2^{\Omega(\sqrt{\log n})};$$

so by Lemma 2, any multiparty protocol for G_k^r has complexity at least $k2^{\Omega(\sqrt{\log n})}$. By Lemma 3, if there good predictor that uses space s, then there is a multiparty protocol for G_r^k with length $O(ks)$, so we must have $s = 2^{\Omega(\sqrt{\log n})}$, as claimed. □

With these parameters, we have a generrator 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 ε-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....