15-855: Intensive Intro to Complexity Theory
Spring 2009

Lecture 16: **Nisan's PRG for small space**

For the next few lectures we will study *derandomization*. In algorithms classes one often sees clever randomized algorithms that perform better than deterministic ones, or which are easier to analyze.

Question: Does randomized computation allow for significant efficiency gains over deterministic computation?

In physical reality we don't have access to independent uniformly random bits, so in practice we use some "pseudorandom bits" (whatever that means). Does this really work, given that we do the analysis based on true random bits? There are notorious examples of papers whose Monte Carlo simulation results were completely wrong because the authors used poor random number generators.

Question: What kinds of randomized algorithms don't change much if true random bits are substituted with some kind of "pseudorandom bits"?

These are the questions we wish to study. Although we cannot unconditionally prove too much about the necessity or lack of same for true randomness, perhaps surprisingly, the evidence points toward the conclusion that *randomness can always be eliminated, with only a small loss of efficiency.*

1 Pseudorandom generators

One can certainly try to derandomize algorithms on a case-by-case basis, but wouldn't it be nice if there was a fixed method which worked simultaneously for a wide variety of algorithms? *Pseudorandom generators* can provide such a method:

Definition 1.1. *A "pseudorandom generator" (PRG) is a (deterministic) map $G : \{0,1\}^\ell \to \{0,1\}^n$, where $n \geq \ell$. Here ℓ is the "seed length" and $n - \ell \geq 0$ is the "stretch". We typically think that $n \gg \ell$ and that G is efficiently computable in some model. If $f : \{0,1\}^n \to \{0,1\}$ is any "statistical test", we say that G "ϵ-fools" f is*

$$|\mathbf{Pr}[f(U_n) = 1] - \mathbf{Pr}[f(G(U_\ell)) = 1]| \leq \epsilon,$$

where U_m denotes a uniformly random string in $\{0,1\}^m$. Here the string U_ℓ is called the "seed". If $\mathcal{C}$ is a class *of tests, we say that G "ϵ-fools $\mathcal{C}$" or is an "ϵ-PRG against $\mathcal{C}$" if G ϵ-fools f for every $f \in \mathcal{C}$.*

In short, G being a PRG against $\mathcal{C}$ means that if you use G to stretch a short truly random seed to a long random string, then that long string "looks uniformly random" to any statistical test in $\mathcal{C}$.

1.1 PRGs for log space

Think about the question of designing a statistical test for deciding if a string of n bits is uniformly random. Probably the first few that you would think of are all computable in L (log space). For example, checking that the number of 0's is in the range, say, $[n/2-\sqrt{n\log n}, n/2+\sqrt{n\log n}]$ is in log space. Or, checking that the longest run of consecutive 1's has length around $\lg n$ is in log space.

Today we will see the following theorem of Nisan:

Theorem 1.2. *(Nisan '90.) Let $\mathcal{C} = \mathsf{SPACE}(S)$. (Think of $S = S(n) = O(\log n)$.) There is a 2^{-S}-generator G against $\mathcal{C}$ which has seed length $\ell = O(S\log n)$ and which is computable in $O(\ell)$ space and* $\mathrm{poly}(\ell)$ *time.*

So we have PRGs for L which have seed length only $O(\log^2 n)$.

Corollary 1.3. *Any language in* BPL *can be decided in space $O(\log^2 n)$ and time $n^{O(\log n)}$.*

Proof. Let $A \in \mathsf{BPL}$ and suppose M is a randomized log-space TM that decides A using $\mathrm{poly}(n)$ random bits (and $\mathrm{poly}(n)$ time). On input $x \in \{0,1\}^n$, think of M_x as a deterministic log-space algorithm whose input is a random string $u \in \{0,1\}^{\mathrm{poly}(n)}$. We know that $\mathbf{Pr}[M_x(u) = 1]$ is either $\geq 2/3$ or $\leq 1/3$, and we are trying to decide which is the case. Construct Nisan's generator G and run $M_x(G(r))$ for all possible seeds $r \in \{0,1\}^{O(\log^2 n)}$; either at least $2/3 - 2^{-O(\log^2)}$ or at most $1/3 + 2^{-O(\log^2)}$ will accept, and this decides $x \in A$. $\square$

Note that this is in some sense *worse* than Savitch's Theorem:

Theorem 1.4. *(Savitch.)* $\mathsf{RL} \subseteq \mathsf{NL} \subseteq \mathsf{DTISP}(n^{O(\log n)}, \log^2 n)$.

However, Nisan's generator has the advantage of working in a **black-box** fashion: whereas the derandomization supplied by Savitch involves looking "at the code" of the RL algorithm, Nisan just says, "hey, just use these pseudorandom bits". Indeed, Nisan subsequently showed that one can improve the above corollary:

Theorem 1.5. *(Nisan '92.) In fact,* $\mathsf{BPL} \subseteq \mathsf{DTISP}(\mathrm{poly}(n), \log^2 n)$*; i.e.,* $\mathsf{BPL} \subseteq \mathsf{SC}$ *("Steve's Class", poly-time, polylog-space).*

This second theorem of Nisan is not much harder than his first theorem; we'll get to it if we have time.

2 Nisan's Generator

2.1 Reduction to automata

Instead of trying to fool log-space machines, it's a bit nicer to try to fool poly-size finite automata.

Definition 2.1. *For the purposes of this lecture, an (m,k)-*automaton *Q is a finite state machine with: a) states $1,\ldots,m$; b) exactly 2^k transitions per state, associated with the strings $\{0,1\}^k$. We identify Q and its transition function, writing $Q(i;x) = j$ to mean Q goes from state i to state j when fed $x \in \{0,1\}^k$.*

We'll do something stronger than fooling $\mathsf{BPSPACE}(S)$ machines which uses n random bits. Instead we'll fool machines that:

- for $k = O(S)$,
- take n blocks of k random bits,
- use space S *between* the blocks,
- and may do any computation within the blocks.

Given any such machine, and an input x, we can build an (m, k)-automaton Q, where the states are the

$$m = 2^{O(S)}$$

configurations, and the transitions are governed by the blocks of k random bits. There is a start state and, WOLOG, a unique, *absorbing* accept state. We can and will assume that n is a power of 2 as well. Our goal then is to build a generator $G : \{0,1\}^\ell \to \{0,1\}^n$, with

$$\ell = O(S \log n) = O(\log m \log n)$$

such that

$$\Pr_{u \in (\{0,1\}^k)^n}[Q^n(\text{start}; u) = \text{acc}] \overset{1/m}{\approx} \Pr_{r \sim \{0,1\}^\ell}[Q^n(\text{start}; G(r)) = \text{acc}].$$

2.2 Closeness of stochastic matrices

Given any (m, k)-automaton Q, let $M(Q)$ denote its "transition matrix",

$$M[i,j] = \Pr_{x \sim \{0,1\}^k}[Q(i; x) = j].$$

We will use the following matrix norm (technically called $\|\cdot\|_\infty$):

Definition 2.2. *Given an $m \times m$ matrix M,*

$$\|M\| = \max_{i \in [m]} \sum_{j=1}^{n} |M_{ij}|;$$

i.e., the largest row sum (when entries are absolute-valued).

Exercise 2.3.

- $\|A + B\| \le \|A\| + \|B\|$.
- $\|AB\| \le \|A\|\|B\|$.

2.3 Mixing lemma

Definition 2.4. *Let $A, B \subseteq \{0,1\}^k$ and let $h : \{0,1\}^k \to \{0,1\}^k$. Define $\alpha = |A|/2^k$, $\beta = |B|/2^k$. We say that h is "ϵ-independent for (A, B)" if*

$$\left|\Pr_{x \sim \{0,1\}^k}[x \in A \wedge h(x) \in B] - \alpha\beta\right| < \epsilon.$$

Lemma 2.5. *("Pairwise independence mixing lemma.") If $h : \{0,1\}^k \to \{0,1\}^k$ is chosen randomly from a pairwise independent hash family $\mathcal{H}_k$, then*

$$\Pr_h[h \text{ not } \epsilon\text{-independent for } (A, B)] \le \frac{\alpha\beta/\epsilon^2}{2^k} \le \frac{(1/\epsilon^2)}{2^k}.$$

Exercise: use Chebyshev, of course.

Proof. For each $x \in A$, let I_x be the 0-1 indicator r.v. for the event $h(x) \in B$ (vis-a-vis the choice of h). It is easy to check that the r.v.'s $\{I_x\}_{x \in A}$ are pairwise independent, since $\mathcal{H}_k$ is. Define

$$C = \sum_{x \in A} I_k = \#\{x \in A : h(x) \in B\}.$$

Notice that once h is chosen, C tells us if h is ϵ-independent for (A, B):

$$h \text{ is } \epsilon\text{-independent for } (A, B) \quad \Leftrightarrow \quad |C/2^k - \alpha\beta| \le \epsilon \quad \Leftrightarrow \quad |C' - \alpha\beta| < \epsilon,$$

where we defined $C' = C/2^k$. By linearity of expectation,

$$\mathop{\mathbf{E}}_h[C'] = (1/2^k) \sum_{x \in A} \mathop{\mathbf{Pr}}_h[h(x) \in B] = (1/2^k) \sum_{x \in A} \beta = \alpha\beta,$$

where we used $\mathbf{Pr}_h[h(x) \in B] = \beta$ for every x, by the pairwise independence of $\mathcal{H}_k$. So the expectation is correct. Our goal is now to use Chebyshev's Inequality.

Since C is a sum of 0-1 pairwise-independent random variables, an exercise from Lecture 5 tells us that

$$\mathop{\mathbf{Var}}_h[C] \le \mathop{\mathbf{E}}_h[C] \quad \Rightarrow \quad \mathop{\mathbf{Var}}_h[C'] = \mathop{\mathbf{Var}}_h[C]/2^{2k} \le \mathop{\mathbf{E}}_h[C]/2^{2k} = \mathop{\mathbf{E}}_h[C']/2^k.$$

Hence by Chebyshev,

$$\mathbf{Pr}[|C' - \alpha\beta| \ge \epsilon]] \le \frac{\mathbf{Var}[C']}{\epsilon^2} \le \frac{\mathbf{E}_h[C']/2^k}{\epsilon^2} = \frac{\alpha\beta/\epsilon^2}{2^k},$$

as needed. □

2.4 Derandomized squaring

Fix an (m, k)-automaton and suppose $M = M(Q)$ is its transition matrix. Let "Q^2" denote the $(m, 2k)$-automaton defined by

$$Q^2(i; x_1, x_2) = Q(Q(i; x_1); x_2).$$

Note that $M(Q^2) = M^2$; i.e., the entries of M^2 are given by

$$M^2[i, j] = \mathop{\mathbf{Pr}}_{x_1, x_2}[Q^2(i; x_1, x_2) = j].$$

If we were asked to estimate one of these entries, the simplest solution would involve using $2k$ random bits. The basic idea in Nisan's generator is that we might be able to do it using only k random bits — if we have the assistance of a good hash function $h : \{0,1\}^k \to \{0,1\}^k$. The idea is, perhaps Q^2 would not notice if instead of giving it independent random x_1 and x_2, we gave it a random x_1 and then $h(x_1)$.

Definition 2.6. *Given an (m, k)-automaton Q and a function $h : \{0,1\}^k \to \{0,1\}^k$, define Q^h to be the "derandomized squared" (m, k) automaton given by*

$$Q_h(i; x) = Q^2(i; x, h(x)).$$

Do Q^2 and Q_h act similarly? We show that if h is chosen at random from a pairwise-independent hash family $\mathcal{H}_k$ of functions $\{0,1\}^k \to \{0,1\}^k$, then this is so:

Lemma 2.7. *Let $M = M(Q)$, so $M^2 = M(Q^2)$, and let $M_h = M(Q_h)$. Then*

$$\mathop{\mathbf{Pr}}_{h\sim\mathcal{H}_k}[\|M^2 - M_h\| \geq \epsilon] \leq \frac{m^7/\epsilon^2}{2^k}.$$

Proof. Fix h, and any entry (i, j). Then

$$\begin{aligned}
\Big|M^2[i,j] - M_h[i,j]\Big| &= \Big|\mathop{\mathbf{Pr}}_{x_1,x_2}[Q(i;x_1,x_2) = j] - \mathop{\mathbf{Pr}}_{x}[Q(i;x,h(x))]\Big| \\
&= \Big|\sum_{p=1}^{m}\mathop{\mathbf{Pr}}_{x_1,x_2}[Q(i;x_1) = p \wedge Q(p;x_2) = j] - \sum_{p=1}^{m}\mathop{\mathbf{Pr}}_{x}[Q(i;x) = p \wedge Q(p;h(x)) = j]\Big| \\
&\leq \sum_{p=1}^{m}\Big|\mathop{\mathbf{Pr}}_{x_1,x_2}[Q(i;x_1) = p \wedge Q(p;x_2) = j] - \mathop{\mathbf{Pr}}_{x}[Q(i;x) = p \wedge Q(p;h(x)) = j]\Big| \\
&= \sum_{p=1}^{m}\Big|\mathop{\mathbf{Pr}}_{x}[Q(i;x) = p]\mathop{\mathbf{Pr}}_{x}[Q(p;x) = j] - \mathop{\mathbf{Pr}}_{x}[Q(i;x) = p \wedge Q(p;h(x)) = j]\Big|.
\end{aligned}$$

The pth term in the summation here depends on how "independent" h is for (A_{ip}, B_{pj}) where

$$A_{ip} = \{x \in \{0,1\}^k : Q(i;x) = p\}, \quad B_{pj} = \{x \in \{0,1\}^k : Q(p;x) = j\}.$$

In particular, suppose h is (ϵ/m^2)-independent for (A_{ip}, B_{pj}) for *every* $i, p, j \in [m]$. Then we would get

$$\Big|M^2[i,j] - M_h[i,j]\Big| \leq m \cdot (\epsilon/m^2) = \epsilon/m$$

for all i, j, and hence $\|M^2 - M_h\| \leq m \cdot (\epsilon/m) = \epsilon$, as required.

By a union bound and the "Pairwise independent mixing lemma", we have that h is (ϵ/m^2)-independent for all (A_{ip}, B_{pj}) except with probability at most $m^3 \cdot (m^4/\epsilon^2)/2^k$, as needed. □

Of course, the next idea is to "derandomized square" again. (You might also ask, where are we going with this? We'll come back to it.) We will not try to get greedy; we will choose two *independent* hash functions, h_1 and h_2. Define the (m, k)-automaton $Q_{h_1h_2} = (Q_{h_1})_{h_2}$; i.e.,

$$Q_{h_1h_2}(i;x) = Q^2_{h_1}(i;x,h_2(x)) = Q^4(i;x,h_1(x),h_2(x),h_1h_2(x)).$$

Naturally, we ask: if $M_{h_1h_2} = M(Q_{h_1h_2})$, is it likely to be close to M^4? We have:

$$\|M^4 - M_{h_1h_2}\| = \|M^4 - M^2_{h_1} + M^2_{h_1} - M_{h_1h_2}\| \leq \|M^4 - M^2_{h_1}\| + \|M^2_{h_1} - M_{h_1h_2}\|. \quad (1)$$

The second term on the RHS of (1) is

$$\|M(Q_{h_1})^2 - M(Q_{h_1h_2})\| = \|M(Q^2_{h_1}) - M((Q_{h_1})_{h_2})\|$$

so it's exactly what's analyzed in Lemma 2.7, just applied to a different (m, k)-automaton: Q_{h_1} rather than Q. Hence for *every* choice of h_1,

$$\mathop{\mathbf{Pr}}_{h_2}[\|M^2_{h_1} - M_{h_1h_2}\| \geq \epsilon] \leq \frac{m^7/\epsilon^2}{2^k}.$$

As for the first term on the RHS of (1), since $A^2 - B^2 = (A-B)A + B(A-B)$ for any matrices, we have

$$\|M^4 - M^2_{h_1}\| = \|(M^2 - M_{h_1})M^2 + M_{h_1}(M^2 - M_{h_1})\| \leq \|M^2 - M_{h_1}\|(\|M^2\| + \|M_{h_1}\|).$$

The first factor on the RHS is again at least ϵ with probability at most $\frac{m^7/\epsilon^2}{2^k}$ over the choice of h_1, by Lemma 2.7. The second factor is always exactly $1+1=2$, because M^2 and M_{h_1} are both transition matrices and hence have all row-sums equal to 1. Hence we can likely bound all of (1) by 3ϵ:

Lemma 2.8.

$$\mathop{\mathbf{Pr}}_{h_1,h_2\sim\mathcal{H}_k}[\|M^4 - M_{h_1h_2}\| \geq 3\epsilon] \leq 2\frac{m^7/\epsilon^2}{2^k}.$$

You probably mostly see where this is going. Let's do one more step. Choose $h_3 \sim \mathcal{H}_k$ independently as well, and define the (m,k)-automaton $Q_{h_1h_2h_3} = (Q_{h_1h_2})_{h_3}$; i.e.,

$$Q_{h_1h_2h_3}(i;x) = Q^2_{h_1h_2}(i;x,h_3(x)) = Q^8(i;x,h_1(x),h_2(x),h_1h_2(x),h_3(x),h_1h_3(x),h_2h_3(x),h_1h_2h_3(x)).$$

We have

$$\|M^8 - M_{h_1h_2h_3}\| \quad = \quad \|M^8 - M^2_{h_1h_2} + M^2_{h_1h_2} - M_{h_1h_2h_3}\| \quad \leq \quad \|M^8 - M^2_{h_1h_2}\| + \|M^2_{h_1h_2} - M_{h_1h_2h_3}\|.$$

The second quantity is, again, at most ϵ with high probability. The first quantity is

$$\|M^8 - M^2_{h_1h_2}\| \leq \|M^4 - M_{h_1h_2}\|(\|M^4\| + \|M_{h_1h_2}\|) \leq 3\epsilon \cdot 2 = 6\epsilon.$$

Hence for the overall bound we get

Lemma 2.9.

$$\mathop{\mathbf{Pr}}_{h_1,h_2,h_3\sim\mathcal{H}_k}[\|M^8 - M_{h_1h_2h_3}\| \geq 7\epsilon] \leq 3\frac{m^7/\epsilon^2}{2^k}.$$

In general, we get

Theorem 2.10.

$$\mathop{\mathbf{Pr}}_{h_1,\dots h_{\lg n}\sim\mathcal{H}_k}[\|M^n - M_{h_1\cdots h_{\lg n}}\| \geq (n-1)\epsilon] \leq (\lg n)\frac{m^7/\epsilon^2}{2^k}.$$

Finally, select $\epsilon = 1/(2mn)$ and $k = C\lg m$ for some large constant C. Then $(n-1)\epsilon \leq 1/(2m)$ and

$$(\lg n)\frac{m^7/\epsilon^2}{2^k} \leq \frac{O(m^9n^2\log n)}{m^C} \leq \frac{m^{12}}{m^C} \leq \frac{1}{2m}$$

by choosing C large enough; here we used $m = 2^{O(S)} \geq n$.

Corollary 2.11. *With $k = O(\log m)$ it holds that*

$$\mathop{\mathbf{Pr}}_{h_1,\dots h_{\lg n}\sim\mathcal{H}_k}[\|M^n - M_{h_1\cdots h_{\lg n}}\| \geq 1/(2m)] \leq 1/(2m).$$

3 The generator itself

We can now state Nisan's generator itself. Recall that we know a simple pairwise independent hash family $\mathcal{H}_k$ (the "$r\cdot x+b$" one) in which a hash function is describable with $2k$ bits and is computable in $O(k)$ time and space. It also has the nice property that picking the $2k$ bits uniformly at random is equivalent to picking a function $h \sim \mathcal{H}_k$ uniformly.

Nisan's generator $G : \{0,1\}^\ell \to \{0,1\}^n$ against $\mathsf{SPACE}(S)$: We set $k = O(\log m) = O(S)$. The seed will consist of a starter string $x \in \{0,1\}^k$, along with $\lg n$ blocks of $O(k)$ bits, each describing a hash function h_i. The total seed length is $\ell = O(S \log n)$, as promised. The generator G's output is defined as

$$(x, h_1(x), h_2(x), h_1h_2(x), h_3(x), h_1h_3(x), h_2h_3(x), h_1h_2h_3(x))$$

in the case $\lg n = 3$, and in general, $G_t(x) = (G_{t-1}(x), h_t(G_{t-1}(x)))$. Note that G is easily computed in $O(\ell)$ time and space. Finally, for any two states i, j in the associated (m, k)-automaton — in particular, $i =$ "start" and $j =$ "accept" — we have:

$$\mathop{\mathbf{Pr}}_{u \sim \{0,1\}^n}[Q^n(i; u) = j] = M^n[i, j],$$

$$\mathop{\mathbf{Pr}}_{v \sim \{0,1\}^\ell}[Q^n(i; G(v)) = j] = \mathop{\mathbf{E}}_{h_1,\ldots,h_{\lg n}}\left[\mathop{\mathbf{Pr}}_{x \sim \{0,1\}^k}[Q_{h_1 \cdots h_{\lg n}}(i; x) = j\right] = \mathop{\mathbf{E}}_{h_1,\ldots,h_{\lg n}}\left[M_{h_1 \cdots h_{\lg n}}[i, j]\right].$$

From Corollary 2.11 we know that

$$|M^n[i, j] - M_{h_1 \cdots h_{\lg n}}[i, j]| \le \|M^n - M_{h_1 \cdots h_{\lg n}}\| \le 1/(2m)$$

except with probability at most $1/(2m)$, in which case the difference is at least bounded by 1. Hence we get

$$\left|\mathop{\mathbf{Pr}}_{u \sim \{0,1\}^n}[Q^n(i; u) = j] - \mathop{\mathbf{Pr}}_{v \sim \{0,1\}^\ell}[Q^n(i; G(v)) = j]\right| \le 1/m = 2^{-O(S)} \le 2^{-S},$$

completing the proof of Nisan's Theorem 1.2.