Spectral Graph Theory and Applications **WS 2011/2012**

Lecture 10: Pseudorandom Generators

Lecturer: Thomas Sauerwald & He Sun

Motivated by the need of making nuclear weapon, Monte Carlo method is introduced by Ulam in 1940s. Closely related is the notion of pseudorandom generators. In 1982, Blum and Micali introduced the notion of a generator which procedures its output in polynomial time such that its output passes a polynomial time test. In the same year, Yao gave another definition of pseudorandom generators, and proved this definition is equal to Blum's definition.

Loosely speaking, pseudorandom generators are defined as efficient *deterministic* algorithms which stretch *short random seeds* into longer pseudorandom sequences. There are three fundamental aspects for pseudorandom generators.

- Efficiency: The generator must be efficient, which means that the pseudorandom generators must produce pseudorandom sequences within polynomial-time. In fact, pseudorandom generators are one kind of *deterministic polynomial-time algorithms*.
- Stretching: The generator is required to stretch its input seed to a longer output sequence. Specifically, the generator stretches an n-bit input into an $\ell(n)$-bit long output, where $\ell(n) > n$. The function ℓ is called the *stretching function* of the generator.
- Pseudorandomness: The generator's output has to look random to any efficient observer. That is, any procedure should fail to distinguish the output of a generator (on a random seed) from a truly random sequence of the same length in the polynomial time. For instance, a procedure could count the number of 0's and 1's and any pseudorandom generator need output almost the same number of 0's and 1's.

Definition 10.1 (Pseudorandom Generators)**.** *A deterministic polynomial-time algorithm G is called a pseudorandom generator if there exists a stretching function $\ell : \mathbb{N} \mapsto \mathbb{N}$, such that the following two probability ensembles, denoted $\{\mathcal{G}_n\}_{n\in\mathbb{N}}$ and $\{U_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable.*

1. *Distribution $\mathcal{G}_n$ is defined as the output of G whose length is $\ell(n)$ on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution U_n is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$, $\ell(n) > n$.*

That is, letting U_m denote the uniform distribution over $\{0,1\}^m$, we require that for any probabilistic polynomial-time algorithm A, any positive polynomial $p(\cdot)$, and for all sufficiently large n, it holds that

$$\Big| \Pr\left[A(G(U_n)) = 1\right] - \Pr\left[A\left(U_{\ell(n)}\right) = 1\right] \Big| < \frac{1}{p(n)}.$$

From the above definition, we know that pseudorandomness is defined in terms of its observer. It is a distribution which cannot be told apart from a uniform distribution by any polynomial-time observer. However, pseudorandom sequences may be distinguished from truly random ones by infinitely powerful observers or more powerful observers. For instance, the pseudorandom sequence that cannot be distinguished from truly random ones by any polynomial-time observer could be distinguished from truly random ones by an exponential-time machine. So pseudorandomness is subjective to the abilities of the observer.

The Formulation of Pseudorandom Generators

The formulation of pseudorandom generators consists of three aspects: (1) The stretching measure of the generators; (2) The class of distinguishers that the generators are supposed to fool, i.e. the class of algorithms that are allowed to distinguish the output of generators and the truly uniform distributions; (3) The resources that generators are allowed to use.

Pseudorandom generators and computational difficulty are strongly related. To show the computational difficulty, we introduce the notion of one-way functions.

One-way functions is the foundation of modern cryptography, and closely related to public-key cryptosystem, pseudorandom generators, and digital signature. Intuitively, one-way functions are the class of functions that are easy to compute and hard to invert. The formal definition is as follows.

Definition 10.2 (One-way Functions). *A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if f satisfies the following two conditions:*

1. *There exists a polynomial-time algorithm A to compute f, i.e. $\forall x : A(x) = f(x)$.*
2. *For all probabilistic polynomial-time algorithms A', polynomials $p(\cdot)$ and sufficiently large n, it holds that*
$$\Pr\left[A'(f(U_n)) = f^{-1} \circ f(U_n)\right] < \frac{1}{p(n)}.$$

We call the algorithm that tries to invert a one-way function or tries to distinguish the output of a pseudorandom generator from a truly random string *an adversary*.

In 1993, Håstad, Impagliazzo, Levin and Luby proved the following definitive theorem: Starting with any one-way function, one can construct a pseudorandom generator.

Theorem 10.3. *Pseudorandom generators exist if and only if one-way functions exist.*

There are a few problems that seem to be one-way in practice and are conjectured to be one-way. A typical example is the discrete logarithm function.

Problem 10.4 (The Discrete Logarithm Problem). *Given a prime modulus p and a generator $g \in \mathbb{Z}_p^*$, for $y = g^x \mod p$ find the index x.*

Before giving the constructions of PRGs, we show that it suffices to construct PRGs with stretching function $\ell(n) = n + 1$.

Theorem 10.5 (amplification of stretch function). *Suppose we have a pseudorandom generator G with a stretch function $n+1$, then for every polynomial $\ell(n) > n$ there exists a pseudorandom generator with stretch function $\ell(n)$.*

Now we turn to discuss pseudorandom generators for bounded computation.

Definition 10.6. *Let M be a randomized* TM *that on input x requires $\ell(|x|)$ random bits. The family $\{g_n\}_{n=1}^{\infty}$ of functions ($g_n : \{0,1\}^{s(n)} \to \{0,1\}^{\ell(n)}$) is an $\varepsilon-$generator for M if for any $x \in \{0,1\}^*$, it holds that*
$$\left| \Pr_{r \in \{0,1\}^{\ell(|x|)}} [M(x,r) = 1] - \Pr_{z \in \{0,1\}^{s(|x|)}} \left[M\left(x, g_{|x|}(z)\right) = 1\right] \right| < \varepsilon.$$

Instead of a family of functions, it is more convenient to view g, the $\varepsilon-$generator for M, as a TM which on input a string of length $s(n)$ outputs a string of length $\ell(n)$. We call z the seed of the PRG.

1 Randomized Logspace TM

There are two different definitions of randomized space bounded TMs based on the manner the random bits are accessed:

- The TM obtains the random bits when they are required.
- The string of random bits is fed as an auxiliary off-line input (on a separate tape) in addition to the regular input to the TM. In this case, the head accessing the random bits on this tape can move back and forth on the tape.

Note that in the case of randomized time bounded computation it is regardless of which convention we observe. For randomized space bounded computation, we consider only TMs of the first kind or equivalently consider TMs of the second kind in which the head on the random tape is restricted in the sense that it can only move right along the tape (i. e., the random tape is one-way read-only tape). We also assume that all randomized space S TMs halt in time less than 2^S.

2 Nisan's Generators

The main task of this lecture is to construct the following PRG.

Theorem 10.7. *[INW94] There exists an n-space-bounded* TM $G : \{0,1\}^{O(\log^2 m)} \to \{0,1\}^m$ *such that for all randomized S-space-bounded* TM M*, G is a 2^{-S}-generator for M, where $m = 2^S$ and $n = O(\log^2 m)$.*

Before discussing the space-bounded PRGs, let us look at the structure of computation tableau of a randomized TM. For an S space-bounded TM M, the computation tableau of M consists of at most 2^S rows and each row corresponds to a configuration of M, as shown in Figure 1.

By the definition of randomized TMs, each step of M's processing requires one random bit and the number of random bits required for M is at most 2^S. For simplicity we assume that the running time of M is 2^S. Consider the tableau divided into two halves with each half requiring random strings r_1 and r_2 respectively each of length $r = 2^S/2$. If r_1 and r_2 are chosen independently, then by definition M can output the correct answer. However, we would like to choose r_1 and r_2 in such a manner that their behavior is not significantly different from the case when they are chosen independently.

Formally, we assume that A_1 and A_2 are the upper and lower halves respectively of the computation tableau of M and, x_1 and x_2 are independently chosen random strings, see Figure 2. Moreover, algorithms A_1 and A_2 are of the following form:

- Algorithm A_1 takes an input x_1 of length r and outputs a string b_1 of length c.
- Algorithm A_2 takes as input the output b_1 of A_1 and another string x_2 of length r and outputs a string b_2 of length c.

What we are in search of is a generator that supplies strings x_1 and x_2 in a fashion better than choosing them independently. For notational brevity, given a function g, we use $g^\ell(z)$ and $g^r(z)$ to express the left half and the right half of the string $g(z)$, i. e., $g(z) = g^\ell(z) \circ g^r(z)$ and $|g^\ell(z)| = |g^r(z)|$. See Figure 3.

Definition 10.8. *A function $g : \{0,1\}^t \to \{0,1\}^r \times \{0,1\}^r$ is defined to be an ε-generator for communication c if for all functions $A_1 : \{0,1\}^r \mapsto \{0,1\}^c$ and $A_2 : \{0,1\}^c \times \{0,1\}^r \mapsto \{0,1\}^c$, we have that*

$$\forall b \in \{0,1\}^c : \qquad \left| \Pr_{x_1,x_2\in\{0,1\}^r} [A_2(A_1(x_1),x_2) = b] - \Pr_{z\in\{0,1\}^t} \left[A_2(A_1(g^\ell(z)),g^r(z)) = b\right] \right| < \varepsilon.$$

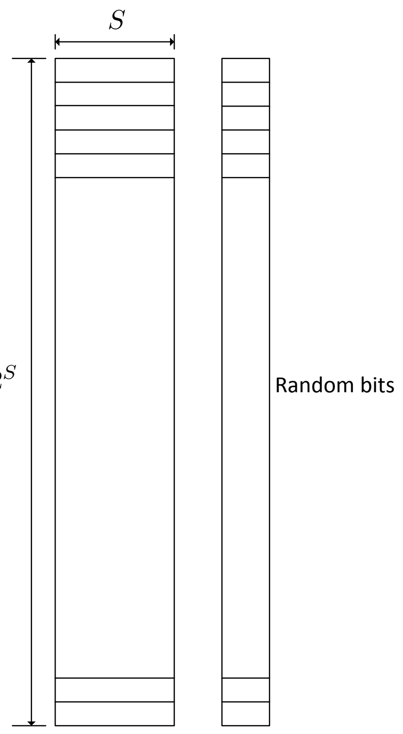

Figure 1: The computation tableau of a space S-bounded TM

$$\begin{tikzcd}
A_1 \arrow[d, "b_1"] & x_1 \arrow[l] \\
A_2 \arrow[d] & x_2 \arrow[l] \\
b_2 &
\end{tikzcd}$$

Figure 2: A_1, A_2 with random inputs

*For simplicity, we call a 2^{-c}-generator for communication c a c-*generator*.*

Let us assume the following lemma holds and use expanders to prove this lemma in the end of the lecture.

Lemma 10.9. *There exists a constant $k > 0$ such that for all r, c, there exists a polynomial time computable c-generator g of the form $g : \{0,1\}^{r+kc} \mapsto \{0,1\}^r \times \{0,1\}^r$.*

Now we break the tableau into several components, called $A_1, A_2, \ldots, A_{2^S}$, and let the output of g be the random strings for every pair of consecutive components (A_{2i-1} and A_{2i}) such that their behavior is not significantly different from using pure random bits, see Figure 4. Moreover, we hope that every application of g causes error at most $1/2^{2S}$. Because we invoke g at most 2^{S-1} times, so the total error is at most $2^{S-1} \cdot 1/2^{2S}$.

Notice that in the above framework there are 2^{S-1} components each of which requires $R+kS$ random bits. We use the same approach to reduce the number of random bits required by every pair of components from $R + kS$ each to $R + 2kS$ total. We perform this operation recursively till there is only one component left. See Figure 5.

Formally let $g_i : \{0,1\}^{R+ikS} \to \{0,1\}^{R+(i-1)kS} \times \{0,1\}^{R+(i-1)kS}$ be an S-generator for

$$\begin{tikzcd}
A_1 \arrow[d, "b_1"] & g^\ell(z) \arrow[l] \\
A_2 \arrow[d] & g^r(z) \arrow[l] \\
b_2 &
\end{tikzcd}$$

Figure 3: A_1, A_2 with inputs from generator

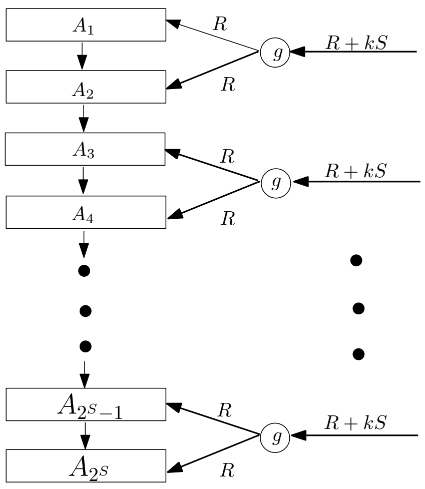

Figure 4: The first level of the recursive construction

$i = 1, \cdots, S$. Define $G_i : \{0,1\}^{R+ikS} \to \{0,1\}^{2^i \cdot R}$ inductively as follows:

$$G_i(z) = \begin{cases} z & i = 0 \\ G_{i-1}\left(g_i^\ell(z)\right) \circ G_{i-1}\left(g_i^r(z)\right) & i > 0 \end{cases}$$

Lemma 10.10. *The PRG G_S runs in space $O\left(R + kS^2\right)$.*

Lemma 10.11. *For all space S-bounded* TM *M, G_S is a 2^{-S}-generator for M.*

Proof. Since each application of g_i incurs an error of at most $1/2^{2S}$, and there are $2^{S-1} + 2^{S-2} + \cdots + 2 + 1 = 2^S - 1$ applications of g_i, it follows by the union bound that the error probability is at most $(2^S - 1)/2^{2S} \leq 1/2^S$. □

3 Proof of Lemma 10.9

Lemma 10.12. *Let $G = (V, E)$ be a d-regular graph with spectral expansion λ. Then for any subsets $S, T \subseteq V$ we have*

$$\left| \frac{|E(S,T)|}{|E|} - \frac{|S|}{|V|} \cdot \frac{|T|}{|V|} \right| \leq \lambda \cdot \sqrt{\frac{|S|}{|V|} \cdot \frac{|T|}{|V|}}.$$

Proof. Apply the Expander Mixing Lemma on the double covering of G. □

Now we prove Lemma 10.9.

Proof. Let $G = (V, E)$ be a $d = 2^{6c}$-regular Ramanujan expander on $|V| = 2^r$ vertices. The generator $g : \{0,1\}^{r+6c} \to \{0,1\}^r \times \{0,1\}^r$ works as follows: On input $z = (x, i) \in \{0,1\}^r \times$

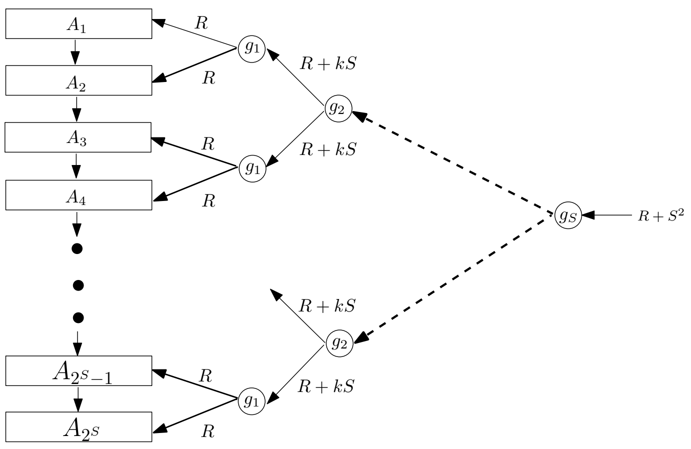

Figure 5: Recursive construction of G_S

$\{0,1\}^d$, output $\left(g^\ell(z), g^r(z)\right) = (x,y)$ where y is the vertex reached by taking the i-th edge out of x.

Let b be any output of the algorithms (A_1, A_2). Fix b. For any $b' \in \{0,1\}^c$, define

$$S_{b'} = \left\{x \in \{0,1\}^r \mid A_1(x) = b'\right\},$$
$$T_{b'} = \left\{x \in \{0,1\}^r \mid A_2(b', x) = b\right\}.$$

Thus for truly random strings x_1 and x_2, the probability that algorithms (A_1, A_2) outputs b is expressed by

$$\Pr_{x_1,x_2}[A_2(A_1(x_1),x_2) = b] = \sum_{b'\in\{0,1\}^c} \Pr[x_1 \in S_{b'} \wedge x_2 \in T_{b'}] = \sum_{b'\in\{0,1\}^c} \frac{|S_{b'}|}{|V|}\cdot\frac{|T_{b'}|}{|V|}.$$

On the other hand, when using the output of g to instead truly random bits, the probability that (A_1, A_2) outputs b becomes

$$\Pr_{z\in\{0,1\}^{r+d}}\left[A_2(A_1(g^\ell(z)), g^r(z)) = b\right] = \sum_{b'\in\{0,1\}^c} \Pr_{x,i}\left[x \in S_{b'} \wedge\ i\text{th edge out of } x \text{ leads to } T_{b'}\right] = \sum_{b'\in\{0,1\}^c} \frac{e(S_{b'},T_{b'})}{|E|}.$$

Therefore

$$\left|\Pr_{x_1,x_2}[A_2(A_1(x_1),x_2) = b] - \Pr_{z\in\{0,1\}^{r+d}}\left[A_2(A_1(g^\ell(z)), g^r(z)) = b\right]\right|$$
$$= \left|\sum_{b'\in\{0,1\}^c}\left(\frac{|S_{b'}|}{|V|}\cdot\frac{|T_{b'}|}{|V|} - \frac{e(S_{b'},T_{b'})}{|E|}\right)\right|$$
$$\le \sum_{b'\in\{0,1\}^c}\left|\frac{|S_{b'}|}{|V|}\cdot\frac{|T_{b'}|}{|V|} - \frac{e(S_{b'},T_{b'})}{|E|}\right|.$$

Because $S_{b'}$ and $T_{b'}$ are subsets of V, by Lemma 10.12 we know that

$$\left|\frac{|S_{b'}|}{|V|}\cdot\frac{|T_{b'}|}{|V|} - \frac{e(S_{b'},T_{b'})}{|E|}\right| \le \lambda\sqrt{\frac{|S_{b'}|}{|V|}\cdot\frac{|T_{b'}|}{|V|}} \le \lambda,$$

where λ is the spectral expansion of G and satisfies $\lambda \le 2 \cdot \sqrt{d-1}/d$. Therefore

$$\begin{aligned}
&\left| \Pr_{x_1,x_2}[A_2(A_1(x_1),x_2)=b] - \Pr_{z\in\{0,1\}^{r+d}}\left[A_2(A_1(g^{\ell}(z)),g^{r}(z))=b\right] \right| \\
\le& \sum_{b'\in\{0,1\}^c} \lambda \\
\le& \sum_{b'\in\{0,1\}^c} \frac{2\cdot\sqrt{d-1}}{d} \\
<& 2^c \cdot \frac{2}{2^{3c}} \\
\le& \frac{1}{2^c},
\end{aligned}$$

which implies that g is a c-generator. □

References

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *STOC*, pages 356–364, 1994.