**CS 710: Complexity Theory** 11/03/2011

# Lecture 17: Space-Bounded Derandomization

Instructor: Dieter van Melkebeek Scribe: Chetan Rao

In the previous lecture, we discussed *Expanders* and also two methods to realize *error reduction* using expanders, namely -

- *Deterministic Error Reduction* - uses a deterministic algorithm ($A_d$). The number of successive runs of $A_d$ required to reduce the error to $\epsilon$ is polynomial in $(1/\epsilon)$ which is larger than performing independent trials ($O(log(1/\epsilon))$.
- *Randomized-efficient Error Reduction* - uses an algorithm ($A_r$) that requires additional random bits. The number of runs of $A_r$ required to reduce the error to $\epsilon$ is logarithmic in $(1/\epsilon)$. In addition to the random bits used by $A_r$, the procedure requires extra random bits logarithmic in $(1/\epsilon)$.

The randomized result was obtained by viewing random bit sequences as vertices of an expander graph and performing a random walk upon choosing a start vertex uniformly at random, and casting a majority vote. The error (probability of majority vote resulting in error) exponentially decreases with the length of the random walk. We also saw a stronger statement based on Chernoff bounds for random walks expander graphs.

In this lecture, we will discuss space-bounded derandomization. We construct a Pseudorandom Generator (PRG) for space-bounded computations based on expanders. The idea is to decrease the required number of seed (random) bits and simulate the algorithm on all possibilities of seed values.

The following section (Section 1) defines Pseudorandom Generators (PRGs) and its various parameters. Section 2 outlines the use of pseudorandom generators in complexity theory. Section 3 concludes with the construction of an efficient PRG for BPL that has seed length of $O((\log n)^2)$.

## 1 Pseudorandom Generators

**Definition 1.** *An $\varepsilon$-PRG for a class $\mathcal{A}$ of algorithms is a sequence $(G_r)_{r=1}^{\infty}$ of deterministic procedures where $G_r : \{0,1\}^{\ell(r)} \to \{0,1\}^r$ such that :*

$$(\forall A \in \mathcal{A})\ (\forall^{\infty} x)\ \|A(x, U_r) - A(x, G_r(U_{\ell(r)}))\|_1 < 2\varepsilon, \tag{1}$$

*where $r$ is the number of random bits $A$ uses on $x$ (and is also the length of the output of $G_r$), $U_n$ denotes $n$ bits taken from the uniform distribution, $\ell(r)$ is the seed length (discussed below) and $\forall^{\infty}$ means "for all except finitely many."*

Note that if $A$ is a decision algorithm, Equation 1 is equivalent to:

$$(\forall^{\infty} x)\ |\Pr[A(x, U_r) \text{ accepts}] - \Pr[A(x, G_r(U_{\ell(r)})) \text{ accepts}]| < \varepsilon \tag{2}$$

There are three important parameters to the above definition.

- *Seed length* $\ell(r)$: the number of random bits required as input to the pseudorandom generator to generate an pseudorandom bit sequence of length $r$. We want this quantity to be small.
- *Error* $\varepsilon$: the deviation from the original randomized algorithm. For example, if the original algorithm has a probability of error $\frac{1}{3}$ and $\varepsilon = \frac{1}{6}$, the probability of error for the new algorithm will be $< \frac{1}{2}$ (by triangle inequality). We want $\varepsilon$ to be small, but it suffices that it be "small enough" given the error reduction techniques discussed in previous lectures.
- *Complexity*: measured in terms of the output length $r$. We want PRGs with low complexity so that using them to generate random bits does not increase the total cost of running a randomized algorithm or its deterministic simulation using the PRG by too much.

## 2 Uses of PRGs

Pseudorandom generators are used to generate a long pseudorandom string from a short uniformly random seed, and allows us to reduce the amount of randomness required to run a randomized algorithm. As a side effect they can reduce the complexity of a deterministic simulation of a randomized algorithm, by explicitly computing the probability of acceptance over the set of all possible PRG seeds $\ell(r) < r$. Namely, if $G$ is a $\frac{1}{6}$-PRG for BPTIME($t$) computable in DTIME($t'$), then

$$\text{BPTIME}(t) \subseteq \text{DTIME}(2^{\ell(t)} \cdot (t'(t) + t)) \tag{3}$$

This is by cycling over all random seeds, running the algorithm on the output of $G$ for each, and outputting the majority answer. For each seed value, the random string must be generated, taking $t'(t)$ time, and the algorithm must be run, for an additional $t$ steps. Since this enumerates all possible seeds and the cumulative error is $< \frac{1}{2}$, a majority vote provides the correct answer.

Similarly, if $G$ is a $\frac{1}{6}$-PRG for BPSPACE($s$) computable in DSPACE($s'$), then

$$\text{BPSPACE}(s) \subseteq \text{DSPACE}(\ell(2^s) + s'(2^s) + s) \tag{4}$$

Given a PRG computable in polynomial time $t'$ with logarithmic seed length $\ell(t)$, BPP $\subseteq$ P. Similarly given a PRG with logarithmic seed length that runs in log space, BPL $\subseteq$ L. Such pseudorandom generators are not known to exist, but this is an approach used to attempt to prove the containments.

## 3 Space-Bounded Derandomization

Although we do not yet know how to construct a log space computable PRG for BPL with $O(\log r)$ seed length, there are nontrivial constructions approaching this goal. We now present a construction based on expanders that yields a PRG for BPL with seed length $O((\log n)^2)$

**Theorem 1.** *There exists an $\varepsilon$-PRG for* BPSPACE($s$) *with*

$$\ell(r) = O(\log \frac{r}{s} \cdot (s + \log \frac{1}{\varepsilon})) \tag{5}$$

*computable in space $O(\ell(r))$.*

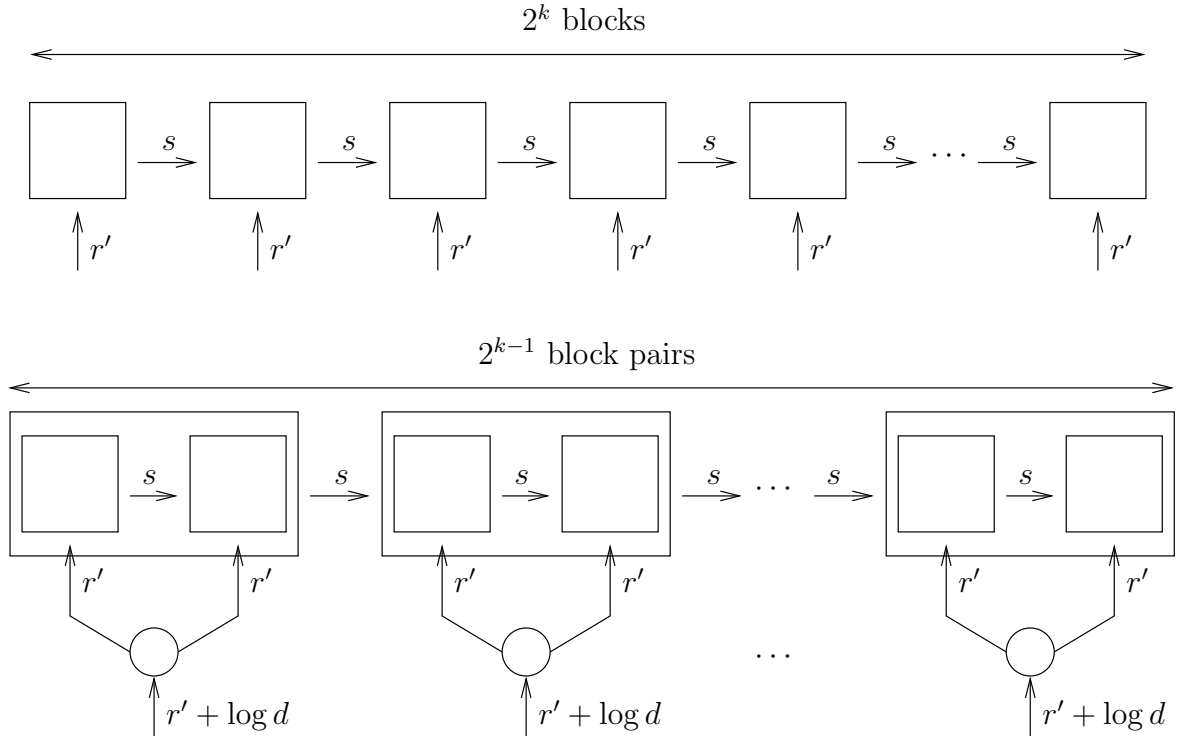


Figure 1: Dividing computation into blocks, with $s$ bits passing between each block. The original computation is shown above, and below it is shown with random bits of adjacent blocks coming from picking adjacent vertices in an expander.

**Corollary 1.** *There is a $\frac{1}{6}$-PRG for BPL with $\ell(r) = O(\log^2 r)$ and computable in space $O(\log^2 r)$.*

Corollary 1 immediately implies that BPL $\subseteq$ DSPACE($\log^2 n$). Earlier, it was already known that BPL $\subseteq$ NC$^2$, but this theorem shows it can be done with PRGs as well. A variation of the PRG can be used in a different way to show that BPL $\subseteq$ DSPACE($\log^{1.5} n$) which is the best known bound.

The idea behind the proof of Theorem 1 is dividing a space-bounded randomized computation into $2^k$ phases. Each phase uses $r'$ random bits, where $r' = \frac{r}{2^k}$. Since the operation of this machine is bounded by space $s$, $s$ bits must pass from phase to phase.

By pairing these blocks and using an expander to produce the random bits for each block, we can reduce the overall level of randomness used by the machine. Consider an expander with degree $d$ and $2^{r'}$ vertices. We let $G_{2r'}$ produce $2r'$ pseudorandom bits by choosing a vertex in the expander at random, then moving to a random neighbor (this is equivalent to selecting an edge at random and using its endpoints). $G_{2r'}$ requires a seed length of $r' + \log d$ random bits for each block pair; if this is $< 2r'$ we have reduced the amount of randomness. This process is diagrammed in 1.

If the expander used is good enough, the output from the modified block pair will not differ greatly from the output of the original. We rely on the expander mixing lemma to prove this.

Call the distribution of input (output resp.) states to a block pair $S_{in}$ ($S_{out}$) and the random inputs to the pair $\rho_{\text{left}}$ and $\rho_{\text{right}}$. There are two distributions to consider for ($\rho_{\text{left}}$,$\rho_{\text{right}}$):

- Random: $U_{2r'}$ - the original randomized input.

- Pseudo-random: $G_{2r'}(U_{r'}, U_{\log d})$ - output from our expander. Note that $G_{2r'}(\rho, \sigma) = (\rho, \sigma\text{-th neighbor of } \rho \text{ in the expander})$.

The following lemma bounds the difference in output distribution between the two scenarios.

**Lemma 1.** *For any distribution $S_{in}$ on $s$ bits where $\lambda$ is the second largest eigenvalue of the expander,*

$$||S_{out}(S_{in}, U_{2r'}) - S_{out}(S_{in}, G_{2r'}(U_{r'}, U_{\log d}))||_1 \leq 2^s \cdot \lambda \tag{6}$$

We prove this lemma in the next lecture, but for now we finish the description of the PRG and the proof of its properties using this lemma. We first want to bound the difference in output distribution of running the algorithm on purely random bits versus running the algorithm by grouping pairs of blocks and producing the random bits from the expander. Consider hybrid distributions, where $D_i$ is the distribution formed by using the random distribution for the first $2i$ blocks, then switching to the pseudo-random distribution for the remainder. Thus $D_{2^{k-1}}$ is perfectly random, and $D_0$ is entirely pseudo-random. The difference between these two distributions is the difference between the randomized algorithm and our pseudorandom version. Also one key point to note is that the difference between any two consecutive hybrid distributions is bounded as follows -

**Claim 1.** $\|D_i - D_{i-1}\|_1 \leq 2^s \cdot \lambda$.

*Proof.* The hybrid distributions $D_i$ and $D_{i-1}$ differs only in the $i^{th}$ position with purely random and pseudorandom sources respectively. From the key lemma, we can bound the error to $\leq 2^s \cdot \lambda$ for the difference in the output of the $i^{th}$ level. If we prove the same error bound for the outputs of $(i+1)^{th}$ level then we are done (by induction). This is true as for any two binary input distributions $X$ and $Y$ ($|X| = |Y| = n$) and a binary deterministic function $f$, if -

$$\begin{aligned} ||X - Y||_1 \leq \delta &\Rightarrow \Pr[\sum_i (x_i - y_i)] < \delta/n \\ &\Rightarrow \Pr[\sum_i (f(x_i) - f(y_i))] < \delta/n \\ &\Rightarrow ||f(X) - f(Y)||_1 \leq \delta \end{aligned}$$

Thus, from the above equations and Lemma 1, we have that $\|D_i - D_{i-1}\|_1 \leq 2^s \cdot \lambda$. □

Hence, from the triangle inequality and Claim 1 we find:

$$\|D_{2^{k-1}} - D_0\|_1 = \left\| \sum_{i=0}^{2^k-1} D_i - D_{i-1} \right\|_1 \leq \sum_{i=1}^{2^{k-1}} \|D_i - D_{i-1}\|_1 \leq 2^{k-1} \cdot 2^s \cdot \lambda \tag{7}$$

This provides a bound on the error introduced by the first step of the derandomization. The amount of randomness has been reduced from $2r'$ for each block pair to $r' + \log d$, a savings of roughly $r'$ as $d$ is constant. This is not a large savings but note that we have reduced our original block chain to an easier instance of the same problem - one with $2^{k-1}$ blocks, each taking $r' + \log d$ random bits. These new computational blocks can be paired, with the $r' + \log d$ random bits being generated by the expander as described above. Note that the expander we use now has more vertices. Pairing blocks recursively (as shown in Figure 2) results in a PRG with the following parameters:

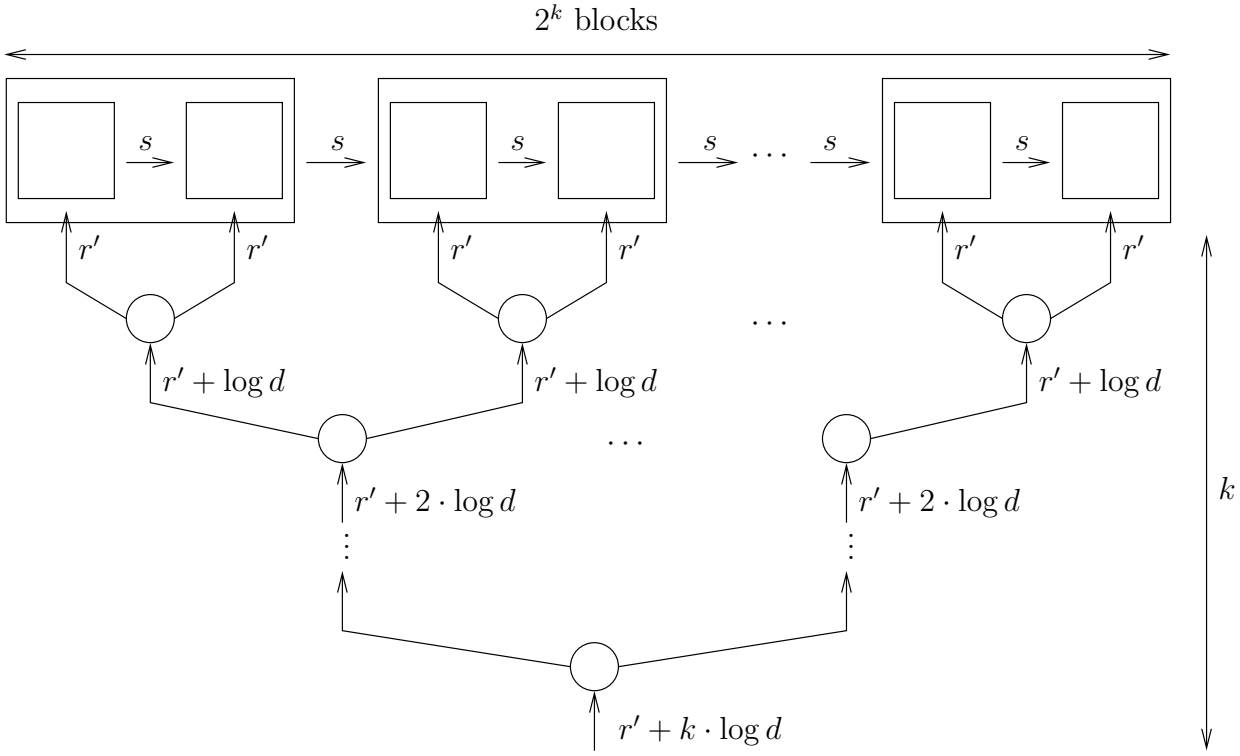


Figure 2: Recursively pairing blocks and applying the expander. $k$ expansions cover the entire computation.

- $\varepsilon < 2^k \cdot 2^s \cdot \lambda$. This bound is found by summing (7) over all levels of recursion.
- $\ell(r) = r' + k \cdot \log d$. Each reduction requires an additional $\log d$ random bits.
- $O(\ell(r))$ space complexity. To compute a given output bit of the PRG, we must compute neighbor relations in a series of expanders. Each of these can be computed in linear space, so the amount of space used at the topmost level dominates. Hence the total space used by the PRG is $O(\ell(r))$.

As defined above $r$ and $r'$ are related through $r' = \frac{r}{2^k}$. The important terms in the parameters for this PRG are $\lambda$ and $d$. Any constant degree expander will have a constant $\lambda$, which will eventually be overshadowed by $2^s$, resulting in $\varepsilon > 1$. To grow $\lambda$ along with $s$ we begin with a constant-degree constant-$\lambda$ expander and raise it to the $t$-th power. Allowing multi-edges in this graph results in a simple expression of the new degree and the second largest eigenvalue in absolute value, namely $\lambda(G^t) = (\lambda(G))^t$, and $d(G^t) = (d(G))^t$. Since we want the error of our PRG to be less than $O(\varepsilon)$ we must satisfy

$$2^{k+s} \cdot \lambda_0^t < O(\varepsilon) \tag{8}$$

We rearrange to derive the value of $t$ that must be used, and plug in $d^t$ as the degree to determine

the seed length

$$t = \Theta(k + s + \log \frac{1}{\varepsilon}) \tag{9}$$

$$\ell(r) = \frac{r}{2^k} + k \cdot \Theta(k + s + \log \frac{1}{\varepsilon}) \cdot \log d \tag{10}$$

We know that $k \leq s$, since there are at most $2^s$ blocks in our construction and each block uses at least one random bit. Therefore, seed length can be given by -

$$\ell(r) = \frac{r}{2^k} + k \cdot \Theta(s + \log \frac{1}{\varepsilon}) \tag{11}$$

The second term grows with $k$ while the first descends. We have remarked before that setting the two terms equal and solving for $k$ gives a result that is minimal to within constant factors. We use $k = \log \frac{r}{s}$. The seed length becomes

$$\ell(r) = O(\log \frac{r}{s} \cdot (s + \log \frac{1}{\varepsilon})) \tag{12}$$

finishing the proof of Theorem 1. All that remains is to prove Lemma 1.

Notice that in our construction each block was treated as a black box. The only connection between blocks was the $s$ bits representing the state of the machine. The algorithm relies on only these $s$ bits being transmitted between blocks, but places no limit on the computations performed by each block individually. This PRG therefore works for any algorithm which can be divided into $2^k$ blocks with limited communication from block to block, even if each block uses unbounded space.

## 4 Next Lecture

In the next lecture, we will prove the Key Lemma (Lemma 1) using the Expander Mixing Lemma. We will also look into similar results in the time bounded setting.

## 5 References

Michael E. Saks and Shiyu Zhou. $\text{BP}_H\text{SPACE}(S) \subseteq \text{DSPACE}(S^{3/2})$. Journal of Computer and System Sciences, 58(2):376403, 1999.

## Acknowledgements

In writing the notes for this lecture, I perused the notes by Jake Rosin for Lecture 14 from the Spring 2007 offering of CS 810, and the notes by Amanda Hittson for Lecture 15 from the Spring 2010 offering of CS 710. The figure credits goes to Jake Rosin.