CS 810: Complexity Theory	2/20/2007
Lecture 14: Space-Bounded Derandomization	
Instructor: Dieter van Melkebeek	Scribe: Jake Rosin

Last time we used expanders to reduce the error of probabilistic algorithms by increasing randomness by only a small amount. Today we attempt the opposite: reducing the amount of randomness required with a bounded increase in error. This lecture focuses on derandomization in a space-bounded setting; in the next lecture we will look at derandomization under time-bounds.

1 Pseudorandom Generators

Definition 1. *An ε-PRG for a class $\mathcal{A}$ of algorithms is a collection $(G_r)_r$ of deterministic procedures where $G_r : \{0,1\}^{\ell(r)} \to \{0,1\}^r$ such that for all $A \in \mathcal{A}$:*

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

Where r is the number of random bits A uses on x, U_n denotes n bits taken from the uniform distribution, and $\forall^\infty$ means "for all x except finitely many."

Note that if A is a decision algorithm, (1) is equivalent to:

$$(\forall^\infty x) \quad \left[|\Pr_{\rho \leftarrow U_r}[A(x,\rho) = 1] - \Pr_{\sigma \leftarrow U_{\ell(r)}}[A(x, G_r(\sigma)) = 1]| < \varepsilon \right] \tag{2}$$

There are three important parameters to the above definition.

- Error ε: 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 $< \frac{1}{2}$. We want ε to be small, but it suffices that it be "small enough" given the amplification techniques discussed in previous lectures.
- Seed length $\ell(r)$: the number of random bits required as input to the pseudorandom generator. We want this small.
- 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 by too much.

2 Uses of PRGs

Pseudorandom generators can be used 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 with $O(\log r)$ seed length, there are nontrivial constructions approaching this goal. We now present a construction based on expanders.

Theorem 1. *There exists an ε-PRG for* BPSPACE(s) *with*

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

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

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

This was already known, due to BPL $\subseteq$ NC^2, but this theorem shows it can be done with PRGs as well. [1]

The idea behind this proof 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 their random bits, 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 Figure 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 ρ_{left} and ρ_{right}. There are two distributions to consider for (ρ_{left},ρ_{right}):

[1] An alternative construction can be used to show that BPL $\subseteq$ DSPACE($\log^{1.5} n$) which is the best known bound.

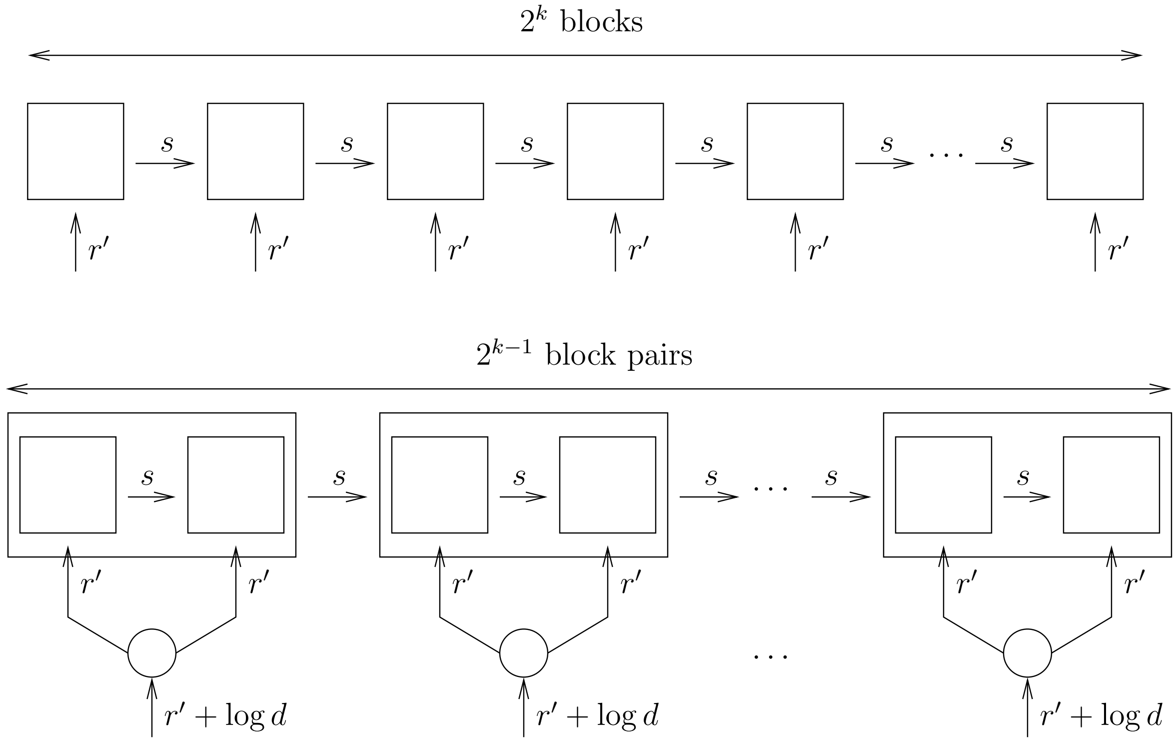

Figure 1: Dividing computation into blocks, with s bits passing between each block. The original computation is shown above, and below it is shown with random bits of adjacent blocks coming from picking adjacent vertices in an expander.

- Random: $U_{2r'}$ - the original randomized input.
- Pseudo-random: $G_{2r'}(U_{r'}, U_{\log d})$ - output from our expander. Note than $G_{2r'}(\rho, \sigma) = (\rho, \sigma$-th neighbor of ρ 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 λ 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 soon prove this lemma, but for now we finish the description and proof of the PRG given the 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. From the triangle inequality and our key lemma we find:

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

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. Pairing blocks recursively (as shown in Figure 2) results in a PRG with the following parameters:

- $\varepsilon < 2^k \cdot 2^s \cdot \lambda$. This bound is found by summing all previous errors.
- $\ell(r) = r' + k \cdot \log d$. 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 λ and d. Any constant degree expander will have a constant λ, which will eventually be overshadowed by 2^s, resulting in $\varepsilon > 1$. To grow λ along with s we begin with a constant-degree constant-λ 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 spectral gap, 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 2ε we must satisfy

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

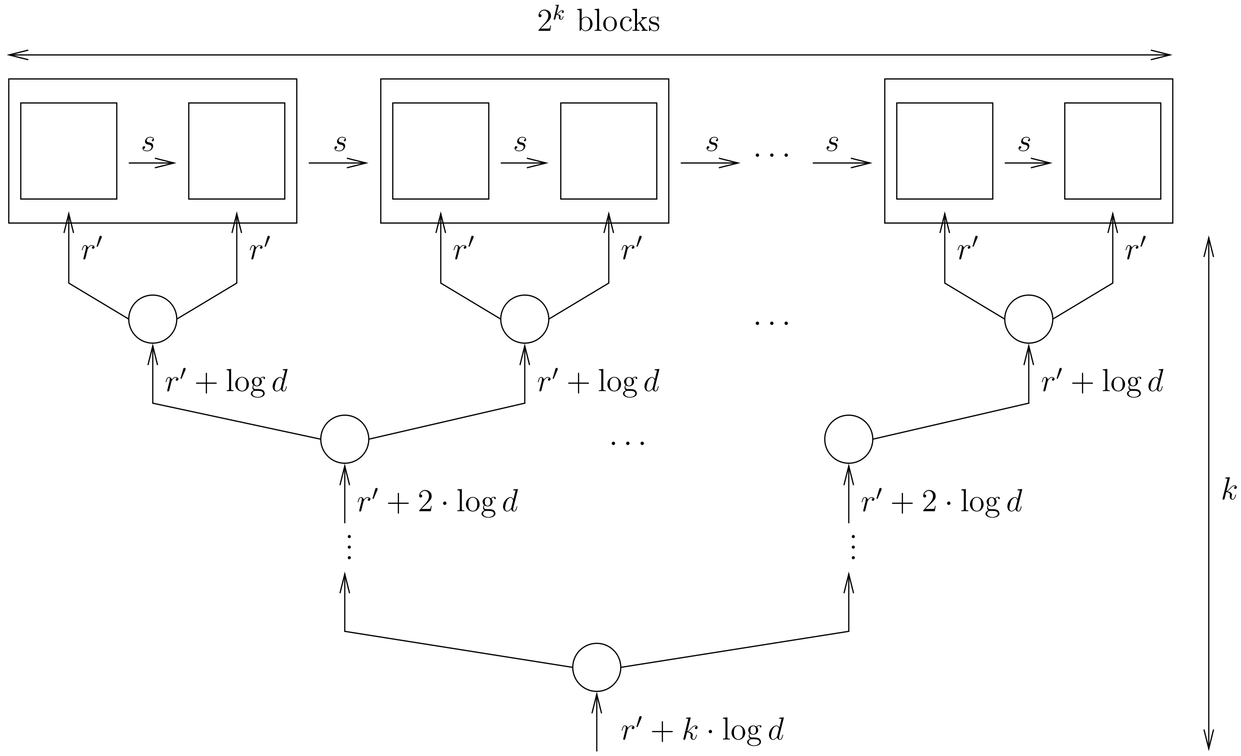

Figure 2: Recursively pairing blocks and applying the expander. k expansions cover the entire computation.

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. Seed length can therefore be defined as

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

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 Proof of Lemma 1

We use the expander mixing lemma for this, and so the proof involves finding the appropriate sets to which to apply the lemma.

Proof. First notice that we can prove the lemma by only considering point distributions. Point distributions place all probability in a single point. Any distribution S_{in} is a convex combination of point distributions $X_1, X_2, ..., X_k$. For a randomized process Z, if we consider the quantity

$$\Delta(S_{in}) = \|Z(S_{in}, U_{2r'}) - Z(S_{in}, G(U_{r'}, U_{\log d}))\|_1,$$

we know by the triangle inequality for 1-norm that $\Delta(S_{in}) \leq \Sigma_k \Delta(X_k)$. Hence, we only need to consider a point distribution s_{in}.

We have defined S_{in} and S_{out} as the distributions of inputs to, and outputs from, respectively, a pair of blocks. We now define S_{mid} as the distribution of states when control passes from the first block of a pair to the second. With this distribution we represent the probability of moving from some input state s_{in} to some output state s_{out} using perfectly random bits as follows

$$\Pr[s_{in} \rightarrow s_{out}] = \sum_{s_{mid}} \Pr[s_{in} \rightarrow s_{mid} \rightarrow s_{out}] \tag{13}$$

$$= \sum_{s_{mid}} \Pr[(\rho_{\text{left}}, \rho_{\text{right}}) \in S \times T] \tag{14}$$

Where S and T are defined as:

$$S = S_{s_{in}, s_{mid}} = \{\rho_{\text{left}} | s_{in} \rightarrow s_{mid}\} \tag{15}$$

$$T = T_{s_{mid}, s_{out}} = \{\rho_{\text{right}} | s_{mid} \rightarrow s_{out}\} \tag{16}$$

In terms of the notation of the expander mixing lemma Equation 14 can be written $\sum_{s_{mid}} \mu(S) \cdot \mu(T)$, as ρ_{left} and ρ_{right} are chosen independently at random.

Under the pseudorandom distribution, the probability of picking an edge between S and T is replaced by $\frac{|E(S,T)|}{dN}$ rather than $\mu(S) \cdot \mu(T)$. Using the expander mixing lemma:

$$\left| \Pr_{(\rho_{\text{left}}, \rho_{\text{right}}) \leftarrow U_{2r'}} [s_{in} \rightarrow s_{out}] - \Pr_{(\rho_{\text{left}}, \rho_{\text{right}}) \leftarrow G_{2r'}(U_{\ell(2r')})} [s_{in} \rightarrow s_{out}] \right| =$$

$$= \left| \sum_{s_{mid}} \mu(S)\mu(T) - \sum_{s_{mid}} \frac{|E(S,T)|}{dN} \right| \leq \sum_{s_{mid}} \lambda \sqrt{\mu(S)\mu(T)}$$

This inequality represents the probability of generating a single fixed s_{out}. To find the difference in probability over S_{out} we take the sum:

$$|S_{out}(s_{in}, \text{random}) - S_{out}(s_{in}, \text{pseudo})|_1 \leq \sum_{s_{out}} \sum_{s_{mid}} \lambda \sqrt{\mu(S)\mu(T)} \tag{17}$$

By Cauchy-Schwartz we can bound this summation by:

$$\sum_{s_{out}} \sum_{s_{mid}} \lambda \sqrt{\mu(S)\mu(T)} \leq \lambda \cdot \sqrt{\sum_{s_{out}} \sum_{s_{mid}} \mu(S)} \sqrt{\sum_{s_{out}} \sum_{s_{mid}} \mu(T)} \tag{18}$$

Given the definition of S provided above, it should be clear that $\sum_{s_{mid}} \mu(S) = 1$. Similarly, the order of summations in the last term can be reversed, and $\sum_{s_{out}} \mu(T) = 1$. We have therefore demonstrated that:

$$|S_{out}(S_{in}, \text{random}) - S_{out}(S_{in}, \text{pseudo})|_1 \leq \lambda \cdot \sqrt{2^s} \cdot \sqrt{2^s} = 2^s \cdot \lambda \tag{19}$$

□

An alternate proof of Theorem 1 using universal hash functions exists, which shows BPSPACE(s) $\subseteq$ DTISP($2^{O(s)}, s^2$). The proof above is more general, as discussed, due to the sole requirement that the computation can be divided into discrete blocks with limited communication.

The alternate proof shows that BPSPACE(s) $\subseteq$ DSPACE(s^2), which we now already know. By exploiting the properties of that construction, however, the inclusion can be improved to BPSPACE(s) $\subseteq$ DSPACE($s^{\frac{3}{2}}$). Although randomness is conjectured to provide no more than a constant factor improvement in space complexity, an overhead of $\sqrt{s}$ is the best known.

5 Next Lecture

In the next lecture we will look at time-bounded derandomization. In a time-bounded setting no non-trivial derandomizations are known; it is possible (though it would be surprising) that BPP = EXP. However, it is possible to perform non-trivial derandomizations under certain reasonable assumptions. As we will see, if there exists a problem in linear exponential time that requires circuits of linear exponential size then BPP = P. In other words, if non-uniformity doesn't help to speed up computations, neither does randomness.