Introduction to Complexity Theory*
Lecture 16: Derandomizing Space-Bounded Computations

Notes taken by Eilon Reshef

May 18, 1999

Summary: This lecture considers derandomization of space-bounded computations. We show that $\mathcal{BPL} \subseteq \mathcal{DSPACE}(\log^2 n)$, namely, any bounded-probability Logspace algorithm can be deterministically emulated in $O(\log^2 n)$ space. We show that $\mathcal{BPL} \subseteq \mathcal{SC}$, namely, any such algorithm can be deterministically emulated in $O(\log^2 n)$ space and (simultaneously) in polynomial time.

1 Introduction

This lecture considers derandomization of space-bounded computations. Whereas current techniques for derandomizing time-bounded computations rely upon unproven complexity assumptions, the derandomization technique illustrated in this lecture stands out in its ability to derive its results exploiting only the pure combinatorial structure of space-bounded Turing machines.

As in previous lectures, the construction yields a pseudorandom generator that "fools" Turing machines of the class at hand, when in our case the pseudorandom generator generates a sequence of bits that looks truly random to any space-bounded machine.

2 The Model

We consider probabilistic Turing machines along the lines of the online model discussed in Lecture 5. A probabilistic Turing machine M has four tapes:

1. A read-only bidirectional input tape.
2. A read-only unidirectional random tape.
3. A read-write bidirectional work tape.
4. A write-only unidirectional output tape.

We consider $\mathcal{BPSPACE}(\cdot)$, the family of *bounded probability* space-bounded complexity classes. These classes are the natural two-sided error counterparts of the single-sided error classes $\mathcal{RSPACE}(\cdot)$, defined in Lecture 7 (Definition **??**).

Formally,

---

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

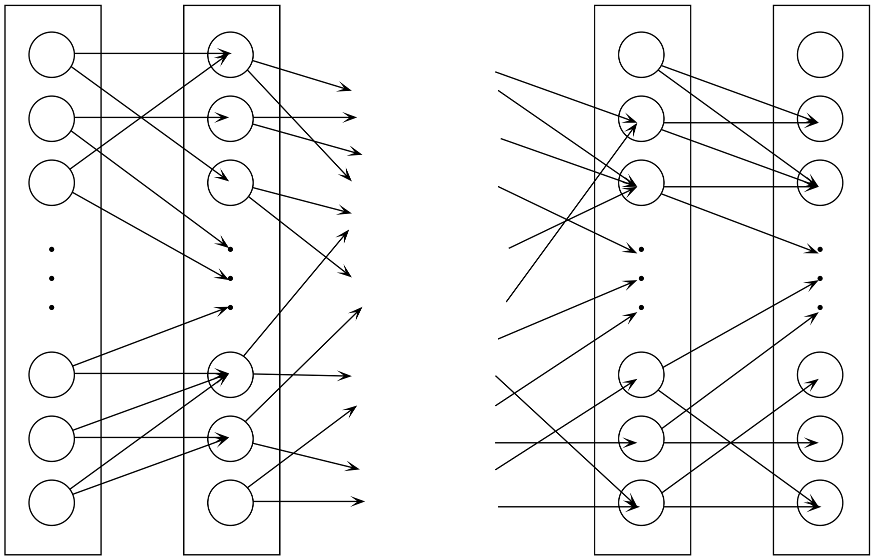

Figure 1: An Execution Graph of a Bounded-Space Turing Machine

Definition 1 *For any function $s(\cdot) : \mathsf{N} \rightarrow \mathsf{N}$, the complexity class $\mathcal{BPSPACE}(s(\cdot))$ is the set of all languages L for which there exists a randomized Turing machine M such that on an input x*

1. *M uses at most $s(|x|)$ space.*
2. *The running time of M is bounded by $\exp(s(|x|))$.*
3. *$x \in L \;\Rightarrow\; \Pr[M(x) = 1] \geq 2/3$.*
4. *$x \notin L \;\Rightarrow\; \Pr[M(x) = 1] < 1/3$.*

Recall that condition (2) is important, as otherwise $\mathcal{NSPACE}(\cdot) = \mathcal{RSPACE}(\cdot) \subseteq \mathcal{BPSPACE}(\cdot)$. As usual, we are interested in the cases where $s(\cdot) \geq \log(\cdot)$. In particular, our techniques derandomize the complexity class $\mathcal{BPL}$, defined as

Definition 2 $\mathcal{BPL} \triangleq \mathcal{BPSPACE}(\log)$

Throughout the rest of the discussion we assume that the problems at hand are decision problems, that all functions are space-constructible, and that all logarithms are of base 2.

3 Execution Graphs

We represent the set of possible executions of a $\mathcal{BPSPACE}(\cdot)$ Turing machine M on an input x, $|x| = n$, as a layered directed graph $G_{M,x}$ (see Figure 1).

A vertex in the i-th layer of $G_{M,x}$ corresponds to a possible configuration of M after it has consumed exactly i random bits, i.e., when the head reading from the random tape is on the i-th cell. Thus, the i-th layer of $G_{M,x}$ corresponds to the set of all such possible configurations. $G_{M,x}$ contains an edge from a configuration vertex in the i-th layer to a configuration vertex in the $(i+1)$-th layer if there is a possible transition between the two configurations.

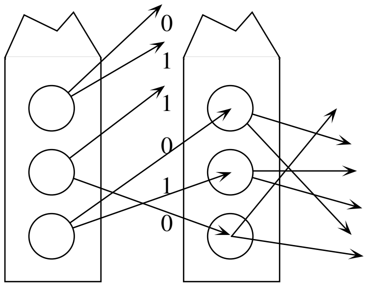

Figure 2: Edges in the Execution Graph $G_{M,x}$

Formally, $G_{M,x} = (V_{M,x}, E_{M,x})$ is defined as follows. For each $i = 1, \ldots, D$, with $D \leq \exp(s(n))$, let the i-th layer $V^i_{M,x}$ be the set of all possible configurations of M given that M has consumed exactly i random bits. The vertex set $V_{M,x}$ is a union of all layers $V^i_{M,x}$. For each vertex $v \in V^i_{M,x}$, the edge set $E_{M,x}$ contains two outgoing directed edges to vertices $f_0(v), f_1(v)$ in $V^{i+1}_{M,x}$, where $f_0(v)$ (resp. $f_1(v)$) corresponds to the configuration M reaches following the sequence of transitions carried out after reading a "0" (resp. "1") bit from the random tape, and until the next bit is read (see Figure 2). For the convenience of the exposition below, assume that each of the two edges is labeled by its corresponding bit, i.e., "0" or "1".

Note that when the location of the head on the random tape is fixed, a configuration of M is fully determined by the contents of the work tape and by the positions of the heads on the work tape and on the input tape. Thus,

$$|V^i_{M,x}| \leq 2^{s(n)} \cdot s(n) \cdot n \leq \exp(s(n)).$$

The initial configuration of M corresponds to a designated vertex $v_0 \in V^0_{M,x}$. Similarly, the set of final vertices $V^D_{M,x}$ is partitioned into the set of accepting configurations V_{ACC} and the set of rejecting configurations V_{REJ}.

A *random walk* on $G_{M,x}$ is a sequence of steps, emanating from v_0, and proceeding along the directed edges of $G_{M,x}$, where in each step the next edge to be traversed is selected uniformly at random. Under this interpretation, the probability that a machine M accepts x is exactly the probability that a random walk emanating from v_0 reaches a vertex in V_{ACC}.

In contrast, a *guided walk* on $G_{M,x}$ with a *guide* R is a sequence of steps, emanating from v_0, and proceeding along the directed edges of $G_{M,x}$, where in the i-th step the next edge to be traversed is determined by the i-th bit of the guide R, i.e., the edge taken is the one whose label is equal to the value of the i-th bit of R.

Let $ACC(G_{M,x}, R)$ denote the event that a guided walk on $G_{M,x}$ with a guide R reaches a vertex in V_{ACC}. In this view,

$$\Pr[M \text{ accepts } x] \quad = \quad \Pr_{R \in_R \{0,1\}^D}[ACC(G_{M,x}, R)],$$

when R is selected uniformly at random from the set $\{0,1\}^D$.

Thus, a language L is in $\mathcal{BPL}$ if there exists a Turing machine M with a space bound of $s(n) = O(\log(n))$ such that for $D(n) = \exp(s(n)) = \mathrm{poly}(n)$,

- Whenever $x \in L$, $\Pr_{R \in_R \{0,1\}^{D(n)}}[ACC(G_{M,x}, R)] \geq 2/3$.

- Whenever $x \notin L$, $\Pr_{R \in_R \{0,1\}^{D(n)}}[ACC(G_{M,x}, R)] < 1/3$.

A (D,W)-*graph* G is a graph that corresponds to the execution of some $s(\cdot)$-space-bounded Turing machine on an input x, with a "depth" (number of layers) of D, $D \leq \exp(s(|x|))$ and a "width" (number of vertices in each layer) of W, $W \leq \exp(s(|x|))$. In the sequel, we present a derandomization method that "fools" any (D,W)-graph by replacing the random guide R with a pseudorandom guide R'.

4 Universal Hash Functions

The construction below is based upon a universal family of hash functions,

$$\mathcal{H}_\ell \;=\; \{h : \{0,1\}^\ell \to \{0,1\}^\ell\}.$$

Recall the following definition:

Definition 3 *A family of functions $\mathcal{H} = \{h : A \to B\}$ is called a* universal family of hash functions *if for every x_1 and x_2 in A, $x_1 \neq x_2$, $\Pr_{h \in_R \mathcal{H}}\left[h(x_1) = y_1 \text{ and } h(x_2) = y_2\right] = \left(\frac{1}{|B|}\right)^2$.*

Note that in our case the family $\mathcal{H}_\ell$ is degenerate, since the functions in $\mathcal{H}_\ell$ map ℓ-bit strings to ℓ-bit strings, and thus do not have any "shrinking" behavior whatsoever.

The construction requires that the functions h in $\mathcal{H}_\ell$ have a succinct representation, i.e., $|\langle h \rangle| = 2\ell$. An example of such a family is the set of all linear functions over $GF(2^\ell)$, namely

$$\mathcal{H}_\ell \;\triangleq\; \{h_{a,b} \;\mid\; a, b \in GF(2^\ell)\},$$

where

$$h_{a,b}(x) \;\triangleq\; ax + b,$$

with $GF(2^\ell)$ arithmetic.

Clearly, $|\langle h_{a,b} \rangle| = 2\ell$, and $h_{a,b}$ can be computed in space $O(\ell)$.

For the purpose of our construction, a hash function h is *well-behaved* with respect to two sets A and B, if it "extends well" to the two sets, i.e.,

$$\left| \Pr_{x \in_R \{0,1\}^\ell}\left[x \in A \text{ and } h(x) \in B\right] - \rho(A) \cdot \rho(B) \right| \leq 2^{-\ell/5},$$

where for any set $S \subseteq \{0,1\}^\ell$, we denote by $\rho(S)$ the probability that a random element x hits the set S, namely,

$$\rho(S) \;\triangleq\; \frac{|S|}{2^\ell} \;=\; \Pr_{x \in \{0,1\}^\ell}[x \in S].$$

The following proposition asserts that for any two sets A and B, almost all of the functions in $\mathcal{H}_\ell$ are well-behaved in the above sense with respect to A and B. Formally,

Proposition 4.1 *For every universal family $\mathcal{H}_\ell$ of hash functions, and for every two sets $A, B \subseteq \{0,1\}^\ell$, all but a $2^{-\ell/5}$ fraction of the $h \in \mathcal{H}_\ell$ satisfy*

$$\left| \Pr_{x \in \{0,1\}^\ell}\left[x \in A \text{ and } h(x) \in B\right] - \rho(A) \cdot \rho(B) \right| \leq 2^{-\ell/5}.$$

5 Construction Overview

We now turn to consider an arbitrary (D,W)-graph G representing the possible executions of an $s(\cdot)$-space-bounded Turing machine M on an input x, where $|x| = n$. We construct a pseudorandom generator $H : \{0,1\}^k \to \{0,1\}^D$ that emulates a truly random guide on G.

Formally,

Definition 4 *A function $H : \{0,1\}^k \to \{0,1\}^D$ is a (D,W)-*pseudorandom generator *if for every (D,W)-graph G,*

$$\left|\Pr_{R\in\{0,1\}^D}[ACC(G,R)] - \Pr_{R'\in\{0,1\}^k}[ACC(G,H(R'))]\right| \leq 1/10.$$

In Section 6, we prove the following theorem:

Theorem 5 *There exists a (D,W)-pseudorandom generator $H(\cdot)$ with $k(n) = O(\log D \cdot \log W)$. Further, $H(\cdot)$ is computable in space linear in its input.*

In particular,

Corollary 6 *There exists a (D,W)-pseudorandom generator $H(\cdot)$ with the following parameters:*

- $s(n) \;=\; \Theta(\log n)$.
- $D(n) \;\leq\; \mathrm{poly}(n)$.
- $W(n) \;\leq\; \mathrm{poly}(n)$.
- $k(n) \;=\; O(\log^2 n)$.

By trying all possible assignments for the seed of $H(\cdot)$, it follows that

Corollary 7 $\mathcal{BPL} \subseteq \mathcal{DSPACE}(\log^2 n)$.

In fact, as will be evident from the construction below, the pseudorandom generator H operates in space $O(\log n)$. However, the space complexity of the derandomization algorithm is dominated by the space used for writing down the seed for $H(\cdot)$.

Note that this result is not very striking, since the same result is known to hold for the single-sided error class $\mathcal{RL}$, as $\mathcal{RL} \subseteq \mathcal{NL} \subseteq \mathcal{DSPACE}(\log^2 n)$. However, as shown below, the construction can be extended to yield more striking results.

6 The Pseudorandom Generator

In this section, we formally describe a (D,W)-pseudorandom generator as defined in Theorem 5. Without loss of generality, we assume $D \leq W$. The pseudorandom generator H is based on the universal family of hash functions $\mathcal{H}_\ell$, and extends strings of length $O(\ell^2)$ to strings of length $D \leq \exp(\ell)$, for $\ell = \Theta(\log|W|)$. The pseudorandom strings cannot be distinguished from truly random strings by any (D,W)-graph.

The input to H is interpreted as the tuple

$$I \;=\; (r, \langle h_1\rangle, \langle h_2\rangle, \ldots, \langle h_{\ell'}\rangle),$$

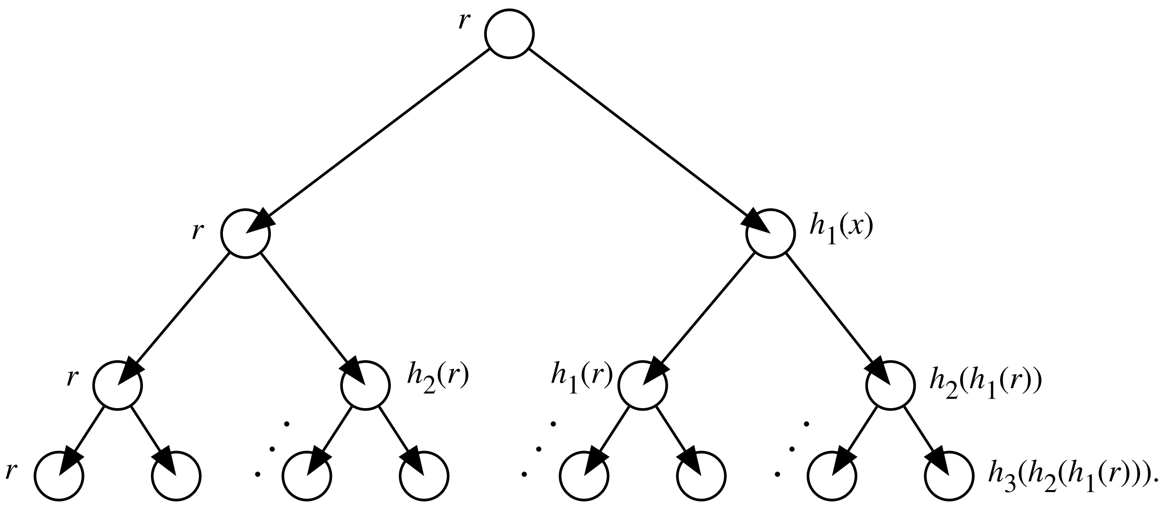

Figure 3: The Computation Tree T of $H(\cdot)$

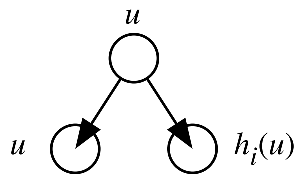

Figure 4: A Node in the Computation Tree of H

where $|r| = \ell$, $h_1, \ldots, h_{\ell'}$ are functions in $\mathcal{H}_\ell$, and $\ell' = \log(D/\ell)$. It can be easily observed that the length of the input I is indeed bounded by $O(\ell^2)$.

Now, given an input I, it may be most convenient to follow the computation of the pseudo-random generator H by considering a computation over a complete binary tree T of depth ℓ' (see Figure 3). The computation assigns a value to each node of the tree as follows. First, set the value of the root of the tree to be r. Next, given that a node located at depth $i-1$ has a value of u, set the value of its left child to u, and the value of its right child to $h_i(u)$ (see Figure 4). Finally, $H(\cdot)$ returns the concatenation of the binary values of the leaves of the tree, left to right.

More formally,

$$H(I) \triangleq \alpha_0 \cdot \alpha_1 \ldots \cdot \alpha_{2^{\ell'}-1},$$

where α_j is defined such that if the binary representation of j is $\tau_1 \cdots \tau_{\ell'}$, then

$$\alpha_j \triangleq h_{\ell'}^{\tau_{\ell'}} \circ \cdots \circ h_1^{\tau_1}(r),$$

where $h_i^1(z) = h_i(z)$ and $h_i^0(z) = z$ for every z.

Yet another way of describing $H(\cdot)$ is recursively as

$$H(r, \langle h_i \rangle, \ldots, \langle h_{\ell'} \rangle) \;=\; H(r, \langle h_{i+1} \rangle, \ldots, \langle h_{\ell'} \rangle) \cdot H(h_i(r), \langle h_{i+1} \rangle, \ldots, \langle h_{\ell'} \rangle),$$

where $H(z) \;=\; z$.

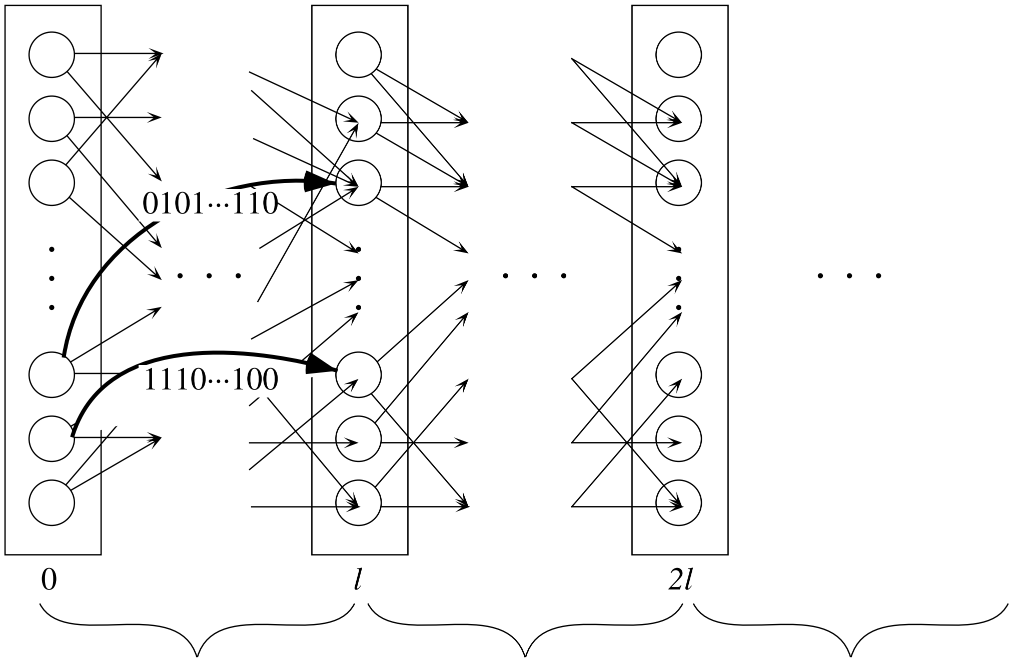

Figure 5: Contracting ℓ Layers

Note that the output of $H(\cdot)$ is composed of exactly $2^{\ell'} = D/\ell$ blocks, each of length ℓ, and hence the length of the output $H(I)$ is indeed D.

7 Analysis

It remains to see that $H(\cdot)$ is indeed a (D, W)-pseudorandom generator.

Theorem 8 (Theorem 5 rephrased) *$H(\cdot)$ is a (D, W)-pseudorandom generator.*

Proof: Consider a (D, W) graph $G_{M,x}$ that corresponds to the possible executions of a prospective distinguisher M on an input x. We show that $H(\cdot)$ is a (D, W)-pseudorandom generator by showing that a guided walk on $G_{M,x}$ using the guide $H(I)$ behaves "similarly" to a truly random walk, where $I = (z, \langle h_1 \rangle, \ldots, \langle h_{\ell'} \rangle)$ is drawn uniformly as a seed of H.

In an initial step, prune layers from $G_{M,x}$, so that only each ℓ-th layer remains, contracting edges as necessary (see Figure 5). Formally, construct a layered multigraph G_0 whose vertex set is the union of the vertex sets $V^0_{M,x}, V^{\ell}_{M,x}, V^{2\ell}_{M,x}, \ldots, V^{D}_{M,x}$, and whose edges correspond to directed paths of length ℓ in $G_{M,x}$. Label each edge in the multigraph G_0 by an ℓ-bit string which is the concatenation of the labels of the edges along the corresponding directed path in $G_{M,x}$. Thus, every multiedge e in G_0 corresponds to a subset of $\{0,1\}^{\ell}$. Clearly, a random walk on $G_{M,x}$ is equivalent to a random walk on G_0, when at each step an ℓ-bit string R is drawn uniformly, and the edge traversed is the edge whose label is R.

The analysis associates $H(\cdot)$ with a sequence of coarsenings. At each such coarsening, a new hash function h_i, uniformly drawn from $\mathcal{H}_{\ell}$, is used to decrease the number of truly random bits needed for the random walk by a factor of 2. After ℓ' such coarsenings, the only truly random bits required for the walk are the ℓ random bits of r, with the additional bits used to encode the hash functions $h_1, \ldots, h_{\ell'}$.

We begin by presenting the first coarsening step. In this step, the random guide $R = (R_1, R_2, \ldots, R_{D/\ell})$ is replaced by a "semi-random" guide $R' = (R_1, h_{\ell'}(R_1), R_3, h_{\ell'}(R_3), \ldots, R_{D/\ell-1}, h_{\ell'}(R_{D/\ell-1}))$, where $h_{\ell'}$ and the R_i's are drawn uniformly at random. Below we show that the semi-random guide behaves "almost like" the truly random guide, i.e., that for some ϵ,

$$|\mathrm{Pr}_R[ACC(G_0, R)] - \mathrm{Pr}_{R'}[ACC(G_0, R')]| < \epsilon.$$

We begin with a technical preprocessing step, removing from G_0 edges whose traversal probability is very small. Formally, let E_{light} denote the set of all "light" edges, i.e., edges (u, v) for which $\mathrm{Pr}_{R_i \in_R \{0,1\}^\ell}[u \to v] < 1/W^2$. Create a graph G'_0 whose vertex set is the same as G_0's, but containing only edges in $E \setminus E_{\text{light}}$. We first show that the removal of the light edges have a negligible effect. Formally,

Lemma 7.1 *For $\epsilon_1 = 2/W$,*

1. $|\mathrm{Pr}_R[ACC(G'_0, R)] - \mathrm{Pr}_R[ACC(G_0, R)]| < \epsilon_1$.
2. $|\mathrm{Pr}_{R'}[ACC(G'_0, R')] - \mathrm{Pr}_{R'}[ACC(G_0, R')]| < \epsilon_1$.

Proof: For the first part, the probability that a random walk R uses an edge in E_{light} is at most $\epsilon_1 = D \cdot (1/W^2) \leq 1/W$, and hence

$$|\mathrm{Pr}_R[ACC(G_0, R)] - \mathrm{Pr}_R[ACC(G'_0), R)]| < \epsilon_1.$$

For the second part, consider two consecutive steps guided by R' along the edges of G_0. The probability that the first step of R' traverses a light edge is bounded by $1/W^2$. By Proposition 4.1 with respect to the sets $\{0,1\}^\ell$ and the set of light edges available for the second step, for all but a fraction of $2^{-\ell/5}$ of the hash functions $h_{\ell'}$, the probability that the second step of R' traverses a light edge is bounded by $1/W^2 + 2^{-\ell/5} \leq 2/W^2$, assuming that the constant for ℓ is large enough. Hence, except for a fraction of $(D/2) \cdot 2^{-\ell/5} < \epsilon_1/2$ of the hash functions, the overall probability that R' traverses a light edge is bounded by $D \cdot (2/W^2) < \epsilon_1/2$. Thus, the overall probability of hitting a light edge is bounded by ϵ_1. ■

It thus remains to show that the semi-random guide R' behaves well with respect to the pruned graph G'_0. Formally, for some ϵ_2 specified below, we show that

Lemma 7.2 $|\mathrm{Pr}_R[ACC(G'_0, R)] - \mathrm{Pr}_{R'}[ACC(G'_0, R')]| < \epsilon_2$.

Proof:

Consider three consecutive layers of G'_0, say $V^0_{M,x}$, $V^\ell_{M,x}$, $V^{2\ell}_{M,x}$ (see Figure 6), and fix a triplet of vertices $u \in V^0_{M,x}$, $v \in V^\ell_{M,x}$, and $w \in V^{2\ell}_{M,x}$ for which the edges (u, v) and (v, w) are in the edge set of G'_0. Let $E_{u,v}$ denote the set of edges in G'_0 connecting u to v, and let $E_{v,w}$ denote the set of edges in G'_0 connecting v to w.

The probability that a random walk emanating from u visits v and reaches w can be written as

$$P_{u-v-w} \;=\; \mathrm{Pr}_{R_1, R_2 \in \{0,1\}^\ell}\left[R_1 \in E_{u,v} \textbf{ and } R_2 \in E_{v,w}\right].$$

Since the graph G'_0 was constructed such that $\mathrm{Pr}_{R_i \in_R \{0,1\}^\ell}[u \to v] \geq 1/W^2$ and $\mathrm{Pr}_{R_i \in_R \{0,1\}^\ell}[v \to w] \geq 1/W^2$, it follows that $P_{u-v-w} \geq 1/W^4$.

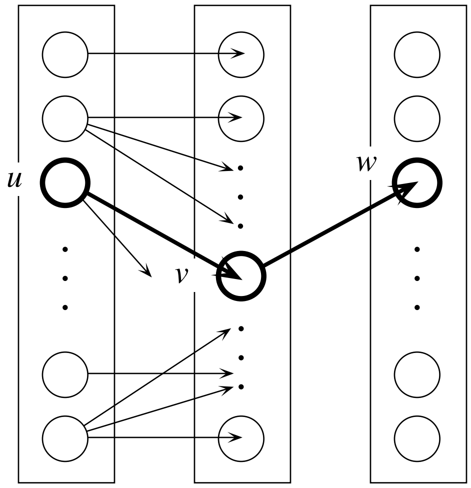

Figure 6: Three Consecutive Layers of G'_0

Now, the crux of the construction is that the above random walk can be replaced by a "semi-random" walk, namely, a walk whose first ℓ steps are determined by a random guide R_1, and whose last ℓ steps are determined by $h_{\ell'}(R_1)$, for some function $h_{\ell'}$ in $\mathcal{H}_\ell$. Given $h_{\ell'}$, the probability that a semi-random walk emanating from u reaches w via v is $P^{h_{\ell'}}_{u-v-w}$, where

$$P^h_{u-v-w} \triangleq \Pr_{R_1 \in \{0,1\}^\ell}[R_1 \in E_{u,v} \textbf{ and } h(R_1) \in E_{v,w}]$$

However, Proposition 4.1 applied with respect to the sets $E_{u,v}$ and $E_{v,w}$ asserts that except for a fraction of $2^{-\ell/5}$ of the hash functions $h_{\ell'}$,

$$\left|P^{h_{\ell'}}_{u-v-w} - P_{u-v-w}\right| \le 2^{-\ell/5}. \tag{1}$$

Thus, except for a fraction of at most $\epsilon_3 = W^4 \cdot 2^{-\ell/5}$ of the hash functions, Equation (1) holds for *every* triplet of vertices u, v and w in every triplet of consecutive layers, i.e.,

$$\forall\, u, v, w \quad \left|P^{h_{\ell'}}_{u-v-w} - P_{u-v-w}\right| \le 2^{-\ell/5}. \tag{2}$$

Next, fix a hash function $h_{\ell'}$ which satisfies Equation (2). The overall probability that a truly random walk starting from u reaches w can be written as

$$P_{u-w} = \sum_v P_{u-v-w},$$

whereas the probability that a semi-random walk starting from u reaches w is

$$P^{h_{\ell'}}_{u-w} = \sum_v P^{h_{\ell'}}_{u-v-w}.$$

Consequently, with a suitable selection of constants, and since $P_{u-w} \ge 1/W^4$,

$$|P^{h_{\ell'}}_{u-w} - P_{u-w}| \le W \cdot 2^{-\ell/5} \le W^5 \cdot 2^{-\ell/5} \cdot P_{u-w} \le 2^{-\ell/10} \cdot P_{u-w} = \epsilon_4 \cdot P_{u-w},$$

for $\epsilon_4 = 2^{-\ell/10}$.

However, once the hash function $h_{\ell'}$ is fixed, every two-hop walk $u - v - w$ depends only on the corresponding ℓ-bit R_i, and hence for every "accepting path", i.e., a path $P \in \{0,1\}^D$ leading from the initial vertex v_0 to an accepting vertex,

$$\left|\Pr[R' = P] - \Pr[R = P]\right| \leq \epsilon_4 \cdot \Pr[R = P].$$

Since the probability of accepting is a sum over all accepting paths,

$$\left|\Pr_{R'}[ACC(G'_0, R')] - \Pr_R[ACC(G'_0, R)]\right| \leq \epsilon_4 \cdot \Pr_R[ACC(G'_0, R)] \leq \epsilon_4. \tag{3}$$

Finally, consider the two events $ACC(G'_0, R)$ and $ACC(G'_0, R')$, where the hash function $h_{\ell'}$ is drawn uniformly at random. The probability that R' hits a "bad" hash function $h_{\ell'}$ is bounded by ϵ_3. Otherwise, Equation (3) holds, and thus the lemma holds for $\epsilon_2 = \epsilon_3 + \epsilon_4$. ■

By the above two lemmas, it follows that the semi-random guide R' behaves "well' in the original graph G_0, i.e.,

Corollary 9 $|\Pr_R[ACC(G_0, R)] - \Pr_{R'}[ACC(G_0, R')]| < \epsilon$, *where* $\epsilon = 2 \cdot \epsilon_1 + \epsilon_2$. ■

Now, to perform another coarsening step, construct a new multigraph G_1 by contracting each pair of adjacent edge sets as follows:

- The vertex set of G_1 is the union of the vertices in the even layers of G.
- Create an edge for every two-hop path taken by the semi-random walk. Formally, for every two adjacent edges (u, v) and (v, w) labeled σ and $h_{\ell'}(\sigma)$ respectively, create an edge (u, w) in G_1, and label it σ.

Reapply Lemma 7.1 and Lemma 7.2 on G_1, this time using a new hash function $h_{\ell'-1}$, yielding a new multigraph G_2, and so on. It thus follows that at each step,

$$\left|\Pr[ACC(G_{i+1}, R^{(i+1)})] - \Pr[ACC(G_i, R^{(i)})]\right| < \epsilon.$$

After ℓ' iterations, the resulting graph $G_{\ell'}$ is a bipartite graph, on which a truly random guide r is applied. Since the above analysis corresponds to the behavior of the pseudorandom generator $H(\cdot)$,

$$\left|\Pr_{I \in_R \{0,1\}^{|I|}}\left[ACC(G, H(I))\right] - \Pr_{R \in_R \{0,1\}^D}\left[ACC(G, R)\right]\right| \quad \leq \quad \ell' \cdot \epsilon \leq 1/10,$$

where $I = (r, \langle h_1 \rangle, \langle h_2 \rangle, \ldots, \langle h_{\ell'} \rangle)$, which concludes the proof. ■

Remark: The analysis requires that the constant for $\ell = \Theta(\log n)$ be large enough. In the underlying computational model, this ensure that the machine M cannot retain even a description of a single function h_i. Otherwise, M could examine the first four blocks of the pseudorandom sequence, i.e., $z, h_{\ell'}(z), z', h_{\ell'}(z')$, and fully determine $h_{\ell'}$ by solving two linear equations with two variables. This would make it possible for M to distinguish between a truly random sequence R and the pseudorandom sequence $H(I)$.

8 Extensions and Related Results

8.1 $\mathcal{BPL} \subseteq \mathcal{SC}$

Whereas Corollary 7 asserts that $\mathcal{BPL} \subseteq \mathcal{DSPACE}(\log^2 n)$, the running time of the straightforward derandomized algorithm is $\Omega(\exp(\log^2(n)))$, and in particular is not polynomial in n.

In this section we consider the complexity class $\mathcal{TS}(t(\cdot), s(\cdot))$, which denotes the set of all languages that can be recognized by a Turing machine whose running time is bounded by $t(\cdot)$ and whose space is (simultaneously) bounded by $s(\cdot)$. In particular, we consider $\mathcal{SC}$ (a.k.a., "Steve's Class"),

Definition 10 $\mathcal{SC} \triangleq \mathcal{TS}(\text{poly}(n), \text{polylog}(n))$.

We state the following theorem:

Theorem 11 $\mathcal{BPL} \subseteq \mathcal{SC}$.

Proof Sketch: Consider a language $L \in \mathcal{BPL}$ and a corresponding Turing machine M for which $L = L(M)$. Now, instead of trying all $O(\log^2(n))$ possible values of the input $I = (r, \langle h_1 \rangle, \ldots, \langle h_{\ell'} \rangle)$ of H to determine whether an input x is in L, perform the following steps:

- "Magically" find a sequence of "good" functions $h_1, \ldots, h_{\ell'}$.
- Emulate M only over all possibilities of r. Since $|r| = \ell$, the emulation can be carried out in time exponential in ℓ, and hence in polynomial time.

To find a sequence of "good" functions, incrementally fix $h_{\ell'}$, $h_{\ell'-1}$, …, h_1. To fix a single h_j, assume all functions $h_{\ell'}, \ldots, h_{j+1}$ were already fixed and stored in memory. Consider all functions h in $\mathcal{H}_\ell$, and test each such h to determine whether Lemma 7.1 and Equation (2) hold. Since the existence of a "good" function h_j is asserted by Theorem 8, it remains to verify that finding such a function can be carried out in time exponential in ℓ (and hence in polynomial time) and in logarithmic space.

To see that this is the case, recall that the pseudorandom generator $H(\cdot)$ can be written recursively as

$$H(r, \langle h_i \rangle, \ldots, \langle h_{\ell'} \rangle) \;=\; H(r, \langle h_{i+1} \rangle, \ldots, \langle h_{\ell'} \rangle) \cdot H(h_i(r), \langle h_{i+1} \rangle, \ldots, \langle h_{\ell'} \rangle),$$

where $H(z) \;=\; z$. Consequently, once the functions $h_{\ell'}, \ldots, h_{j+1}$ are fixed, every single probability P_{u-v-w} can be computed directly simply by exhaustively considering all possible random guides R (of length 2ℓ). Similarly, given a candidate hash function $h_{\ell'}$, every single probability $P^{h_{\ell'}}_{u-v-w}$ can also be computed directly by exhaustively considering all semi-random guides R'. Hence, to determine whether Equation (2) holds, simply compare P_{u-v-w} and $P^{h_{\ell'}}_{u-v-w}$ for every triplet of vertices u, v and w in adjacent layers. Testing whether Lemma 7.1 holds can be carried out in a similar manner. Clearly, each such test can be carried out in time exponential in ℓ, and since the number of candidate functions $h_{\ell'}$ is also exponential in ℓ, the overall running time is exponential in ℓ. Further, testing a single function h can be carried out in space linear in ℓ, and hence the overall space complexity is dominated by the space needed to store the functions $h_{\ell'}, \cdots, h_1$, i.e., by $O(\ell^2)$. ■

Remark: The edge set of the i-th multigraph G_i depends upon the hash functions drawn in previous steps. Thus, although "almost all" functions h in $\mathcal{H}_\ell$ would satisfy Equation (2), one cannot consider fixing a function h_j before committing on the functions $h_{\ell'}, \ldots, h_{j+1}$.

8.2 Further Results

Below we state, without a proof, two related results:

Theorem 12 $\mathcal{BPL} \subseteq \mathcal{DSPACE}(\log^{1.5} n)$.

Theorem 13 (Informal) *Every random computation that can be carried out in polynomial time and in linear space can also carried out in polynomial time and linear space, but using only a linear amount of randomness.*