Electronic Colloquium on Computational Complexity, Report No. 35 (2010)

Pseudorandom Generators for Regular Branching Programs

Mark Braverman* Anup Rao† Ran Raz‡ Amir Yehudayoff§

Abstract

We give new pseudorandom generators for *regular* read-once branching programs of small width. A branching program is regular if the in-degree of every vertex in it is (0 or) 2. For every width d and length n, our pseudorandom generator uses a seed of length $O((\log d + \log\log n + \log(1/\epsilon))\log n)$ to produce n bits that cannot be distinguished from a uniformly random string by any regular width d length n read-once branching program, except with probability ϵ.

We also give a result for general read-once branching programs, in the case that there are no vertices that are reached with small probability. We show that if a (possibly non-regular) branching program of length n and width d has the property that every vertex in the program is traversed with probability at least γ on a uniformly random input, then the error of the generator above is at most $2\epsilon/\gamma^2$.

1 Introduction

This paper is about quantifying how much additional power access to randomness gives to space bounded computation. The main question we wish to answer is whether or not randomized logspace is the same as logspace. This project has a long history [AKS87, BNS89, Nis92, Nis94, NZ96, SZ99] (to mention a few), showing how randomized logspace machines can be simulated by deterministic ones. Savitch [Sav70] showed that nondeterministic space S machines can be simulated in deterministic space S^2, implying in particular that $RL \subseteq L^2$. Subsequently Saks and Zhou showed that $BPL \subseteq L^{3/2}$ [SZ99], which is currently the best known bound on the power of randomization in this context.

One way to simulate randomized computations with deterministic ones is to build a *pseudorandom generator*, namely, an efficiently computable function $g : \{0,1\}^s \to \{0,1\}^n$ that can stretch a short uniformly random seed of s bits into n bits that cannot be distinguished from uniform ones by small space machines. Once we have such a generator, we can obtain a deterministic computation by carrying out the computation for every fixed setting of the seed. If the seed is short enough, and the generator is efficient enough, this simulation remains efficient.

The computation of a randomized Turing machine with space S that uses R random bits can be modeled by a branching program of width 2^S and length R. Complementing Savitch's result above, Nisan [Nis92] showed that there is a pseudorandom generator that can stretch $O(\log^2 n)$ bits to get n bits that are pseudorandom for branching programs of width n and length n. Subsequently, there were other constructions of pseudorandom generators, [NZ96, INW94, RR99], but no better

*Microsoft Research New England. Email: mbraverm@cs.toronto.edu.
†University of Washington. Email: anuprao@cs.washington.edu.
‡Weizmann Institute of Science. Email: ran.raz@weizmann.ac.il.
§Institute for Advanced Study. Email: amir.yehudayoff@gmail.com.

ISSN 1433-8092

seed length for programs of width n and length n was obtained. In fact, no better results were known even for programs of width 3 and length n.

In this work, we give new pseudorandom generators for *regular branching programs*. A branching program of width d and length n is a directed graph with nd vertices, arranged in the form of n layers containing d vertices each. Except for vertices in the final layer, every vertex in the program has two outgoing edges into the next layer, labeled 0 and 1. The program has a designated start vertex in the first layer and an accept vertex in the final layer. The program accepts an input $x \in \{0,1\}^n$ if and only if the path that starts at the start vertex and picks the outgoing edge for the i'th layer according to the input bit x_i ends at the accept vertex. The program is regular if every vertex has in-degree 2 (except for vertices in the first layer that have in-degree 0). The main result of this work is a pseudorandom generator with seed length $O((\log d + \log\log n + \log(1/\epsilon))\log n)$ and error ϵ, for regular branching programs of length n and width d.

We observe that regular programs are quite powerful: Every circuit in NC_1 can be simulated by a regular width 5 (multiple read) branching program of polynomial size, by Barrington's celebrated result [Bar89]. The restriction that the random bits are read only once is natural if one view the random bits as coin-flips (i.e., the previous bit is erased once the coin is flipped again) rather than a random tape that can be traversed back and fourth. We note, however, that our result does not give any derandomization result for NC_1, since Barrington's reduction does not preserve the read-once property.

Our result also gives a generalization of an ϵ-biased distribution for arbitrary groups. An ϵ-biased distribution is a distribution on bits $Y_1, \ldots, Y_n$ such that for every $g_1, \ldots, g_n \in \mathbb{Z}_2$, the distribution of $\sum_i Y_i \cdot g_i$ is ϵ-close to the distribution of $\sum_i U_i \cdot g_i$, where $U_1, \ldots, U_n$ are uniformly random bits and the sum is taken modulo 2. Saks and Zuckerman showed that ϵ-biased distributions are also pseudorandom for width 2 branching programs [SZ]. Today, we know of several explicit constructions of ϵ-biased distributions using only $O(\log n)$ seed length [NN93, AGHP92], which have found a large number of applications in computer science. Our distribution gives a generalization of this object to arbitrary groups: for $Y_1, \ldots, Y_n$ as in our construction, and a group G of size d, our construction guarantees that tests of the form $\prod_i g_i^{Y_i}$ cannot distinguish the Y_i's from being uniform.

1.1 Techniques

Our construction builds on the ideas of a line of pseudorandom generators [Nis92, NZ96, INW94]. Indeed, the construction of our generator is the same as in previous works and our improvements come from a more careful analysis. Previous works gave constructions of pseudorandom generators based on the use of *extractors*. Here, an extractor is an efficiently computable function $\mathsf{Ext} : \{0,1\}^r \times \{0,1\}^{O(k+\log(1/\epsilon))} \to \{0,1\}^r$ with the property that if X is any random variable with min-entropy at least $r-k$, and Y is a uniformly random string, the output $\mathsf{Ext}(X,Y)$ is ϵ-close to being uniform.

Earlier works [Nis92, NZ96, INW94] gave the following kind of pseudorandom generator for branching programs of length n (assume for simplicity that n is a power of 2). For a parameter s, we define a sequence of generators[1] $G_0, \ldots, G_{\log n}$. Define $G_0 : \{0,1\}^s \to \{0,1\}$ as the function outputting the first bit of the input. For every $i > 0$, define $G_i : \{0,1\}^{i \cdot s} \times \{0,1\}^s \to \{0,1\}^{2^i}$ as

$$G_i(x,y) = G_{i-1}(x) \circ G_{i-1}(\mathsf{Ext}(x,y)),$$

[1]The logarithms in this paper are always of base 2.

where $\circ$ means concatenation.

The function $G_{\log n}$ maps a seed of length $s \cdot (\log n + 1)$ to an output of length n. The upper bound on the errors of the generators is proved by induction on i. Let us denote the error of the i'th generator ϵ_i. For the base case, the output of the generator is truly uniform, so $\epsilon_0 = 0$. For the general case, the idea is that although the second half of the bits is not independent of the first half, conditioned on the vertex reached in the middle, the seed x has roughly $i \cdot s - \log d$ bits of entropy (where d is the width of the program). Thus, if $s \geq \Omega(\log d + \log(1/\epsilon))$, the seed for the second half is ϵ-close to uniform, even when conditioned on this middle vertex. Thus, the total error can be bounded by $\epsilon_i \leq (\epsilon_{i-1}) + (\epsilon_{i-1} + \epsilon) = 2\epsilon_{i-1} + \epsilon$, giving $\epsilon_{\log n} = O(n\epsilon)$. In order to get a meaningful result, ϵ must be bounded by $1/n$, which means that, according to this analysis, the seed length of the generator must be at least $\Omega(\log^2 n)$.

In our work, we give a more fine-grained analysis of this construction, that gives better parameters for regular branching programs. To illustrate our ideas, let us consider two extreme examples. First, suppose we have a branching program that reads 2^i bits, and the final output of the program does not depend on the second half of the bits: the vertex at the $2^{i-1}+1$ layer determines the final vertex that the program reaches. For such a program, we can bound the error by $\epsilon_i \leq \epsilon_{i-1}$. This is because only the distribution on the $2^{i-1}+1$ layer is relevant. On the other hand, suppose we had a program where only the last 2^{i-1} bits of input are relevant, in the sense that every starting vertex in the middle layer has the same probability of accepting a uniformly random 2^{i-1} bit string. In this case, we can bound the error by $\epsilon_i \leq \epsilon_{i-1} + \epsilon$.

In general, programs are a combination of these two situations. The program has d possible states at any given time, and intuitively, if the program needs to remember much information about the first 2^{i-1} bits, then it cannot store much information about the next 2^{i-1} bits. This is the fact that we shall exploit. In order to do so, we shall need to formalize how to measure the information stored by a program.

For every vertex v in the program, we label the vertex by the number $q(v)$, which is the probability that the program accepts a uniformly random string, starting at the state v. To every edge (u, v) in the program, we assign the weight $|q(v) - q(u)|$. Our measure of the information in a segment of the program is the total weight of all edges in that segment. Checking with our examples above, we see that if the total weight of the second half of the program is 0, then the middle layer of the program must determine the output. On the other hand, if all vertices in the middle layer have the same value of $q(v)$, then the weight of all edges in the first half must be 0. A key observation is that if the input bits are replaced with bits that are only ϵ-close to uniform, then the outcome of the program can change by at most ϵ times the weight of the program.

The proof proceeds in two steps. In the first step, we show via a simple combinatorial argument that the total weight of all edges in a regular branching program of width d is bounded by $O(d)$. To argue this, we use regularity; for non-regular programs, the weight can grow with n. In the second step, we prove by induction on i that $\epsilon_i \leq O(i \cdot \epsilon \cdot d \cdot \mathsf{weight}_P)$, where here weight_P is the total weight of all edges in the program P. If $\mathsf{weight}_P = \mathsf{weight}_Q + \mathsf{weight}_R$, where Q, R are the first and second parts of the program, the contribution to ϵ_i of the first half is at most $O((i-1) \cdot \epsilon \cdot d \cdot \mathsf{weight}_Q)$ by induction. If the seed to the second half was truly uniform, the contribution of the second half would be at most $O((i-1) \cdot \epsilon \cdot d \cdot \mathsf{weight}_R)$. Instead, it is only ϵ-close to uniform, which contributes an additional error term of $O(\epsilon \cdot d \cdot \mathsf{weight}_R)$. Summing the three terms proves the bound we need.

The total error of the generator is thus bounded by $O(\log n \cdot \epsilon \cdot d \cdot \mathsf{weight}_P)$. Now we only need to set ϵ to be roughly $1/(d^2 \log n)$ to get a meaningful result. This reduces the seed length of the

generator to $O((\log d + \log\log n)\log n)$.

2 Preliminaries

Branching Programs

For an integer n, denote $[n] = \{1, 2, \ldots, n\}$. Fix two integers n, d and consider the set of nodes $V = [n] \times [d]$. For $t \in [n]$, denote $V_t = \{(t, i)\}_{i \in [d]}$. We refer to V_t as *layer* t of V.

A *branching program* of length n and width d is a directed (multi-) graph with set of nodes $V = [n] \times [d]$, as follows: For every node $(t, i) \in V_1 \cup \ldots \cup V_{n-1}$, there are exactly 2 edges going out of (t, i) and both these edges go to nodes in V_{t+1} (that is, nodes in the next layer of the branching program). One of these edges is labeled by 0 and the other is labeled by 1. Without loss of generality, we assume that there are no edges going out of V_n (the last layer of the branching program). A branching program is called *regular* if for every node $v \in V_2 \cup \ldots \cup V_n$, there are exactly 2 edges going into v (note that we do not require that the labels of these two edges are different).

Paths in the Branching Program

We will think of the node $(1, 1)$ as the starting node of the branching program, and of $(n, 1)$ as the accepting node of the program. For a node $v \in V_1 \cup \ldots \cup V_{n-1}$, denote by $\text{next}_0(v)$ the node reached by following the edge labeled by 0 going out of v, and denote by $\text{next}_1(v)$ the node reached by following the edge labeled by 1 going out of v.

A string $x = (x_1, \ldots, x_r) \in \{0,1\}^r$, for $r \leq n-1$, defines a path in the branching program $\text{path}(x) \in ([n] \times [d])^{r+1}$ by starting from the node $(1, 1)$ and following at step t the edge labeled by x_t. That is, $\text{path}(x)_1 = (1, 1)$ and for every $t \in [r]$, $\text{path}(x)_{t+1} = \text{next}_{x_t}(\text{path}(x)_t)$.

For a string $x \in \{0,1\}^{n-1}$, and a branching program B (of length n), define $B(x)$ to be 1 if $\text{path}(x)_n$ is the accepting node, and 0 otherwise.

Remark 1. *As the definitions above indicate, for the rest of this paper a branching program is always read-once.*

Distributions over $\{0,1\}^n$

For a distribution D over $\{0,1\}^n$, we write $x \sim D$ to denote that x is distributed according to D. Denote by U_k the uniform distribution over $\{0,1\}^k$. For a random variable z and an event A, denote by $z|A$ the random variable z conditioned on A. For a function ν, denote by $|\nu|_1$ its L^1 norm. We measure distances between distributions and functions using the L^1 distance.

3 Evaluation Programs

An *evaluation program* P is a branching program, where every vertex v is associated with a value $q(v) \in [0, 1]$, with the property that if the outgoing edges of v are connected to v_0, v_1, then

$$q(v) = \frac{q(v_0) + q(v_1)}{2}. \tag{1}$$

Every branching program induces a natural evaluation program by labeling the last layer as

$$q((n,i)) = \begin{cases} 1 & \text{if } i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

and then labeling each layer inductively by Equation (1).

Given $x \in \{0,1\}^r$, and an evaluation program P, we shall write $\mathsf{val}_P(x)$ (or simply $\mathsf{val}(x)$, when P is clear from context) to denote the quantity $q(\mathrm{path}(x)_{r+1})$, namely, the value $q(v)$ of the vertex v reached by starting at the start vertex and taking the path defined by x. We shall write $\mathsf{val}(x,y)$ to denote the value obtained by taking the path defined by the concatenation of x, y.

We shall use the following three simple propositions.

Proposition 2. *If U is the uniform distribution on r bit strings, $\mathbb{E}_{u\sim U}[\mathsf{val}(x,u)] = \mathsf{val}(x)$.*

We assign a weight of $|q(u) - q(v)|$ for every edge (u,v) of the evaluation program. The *weight* of the evaluation program P is the sum of all the weights of edges in the program. We denote this quantity by weight_P.

Proposition 3. *Let X, Y be two distributions on r bit strings, and P be an evaluation program. Then*

$$\left| \mathop{\mathbb{E}}_{x\sim X}[\mathsf{val}_P(x)] - \mathop{\mathbb{E}}_{y\sim Y}[\mathsf{val}_P(y)] \right| \leq \frac{|X - Y|_1 \cdot \mathsf{weight}_P}{2}.$$

Proof. Let $\mathsf{val}_{\mathsf{max}}$ denote the maximum value of $\mathsf{val}(b_1)$ and $\mathsf{val}_{\mathsf{min}}$ denote the minimum value of $\mathsf{val}(b_2)$ over all choices of $b_1, b_2 \in \{0,1\}^r$. Assume that $\mathsf{val}_{\mathsf{max}} \neq \mathsf{val}_{\mathsf{min}}$ (otherwise the proof is trivial). Let v_{max} be the vertex reached by a string b_1 for which the maximum is attained, and let $v_{\mathsf{min}} \neq v_{\mathsf{max}}$ be the vertex reached by a string b_2 for which the minimum is attained. Let $\gamma_{\mathsf{max}}, \gamma_{\mathsf{min}}$ be two edge disjoint paths in the program starting at some node v and ending at $v_{\mathsf{max}}, v_{\mathsf{min}}$, respectively. Such paths must exist, since $v_{\mathsf{max}}, v_{\mathsf{min}}$ are both reachable from the start vertex of the program. By the triangle inequality, $\mathsf{val}_{\mathsf{max}} - \mathsf{val}_{\mathsf{min}}$ is bounded by the total weight on the edges of these paths, which implies

$$\mathsf{val}_{\mathsf{max}} - \mathsf{val}_{\mathsf{min}} \leq \mathsf{weight}_P.$$

Let $x \sim X$ and let $y \sim Y$. Let B denote the set $\{b \in \{0,1\}^r : \Pr[x = b] \geq \Pr[y = b]\}$. Observe that

$$\sum_{b\in B} \Pr[x = b] - \Pr[y = b] = \sum_{b\notin B} \Pr[y = b] - \Pr[x = b] = |X - Y|_1/2.$$

Without loss of generality, assume that $\mathbb{E}_{x\sim X}[\mathsf{val}_P(x)] \geq \mathbb{E}_{y\sim Y}[\mathsf{val}_P(y)]$. We bound

$$\begin{aligned}
&\mathop{\mathbb{E}}_{x\sim X}[\mathsf{val}(x)] - \mathop{\mathbb{E}}_{y\sim Y}[\mathsf{val}(y)] \\
&\qquad = \sum_{b\in\{0,1\}^r} \Pr[x = b] \cdot \mathsf{val}(b) - \Pr[y = b] \cdot \mathsf{val}(b) \\
&\qquad \leq \sum_{b\in B} (\Pr[x = b] - \Pr[y = b]) \cdot \mathsf{val}_{\mathsf{max}} + \sum_{b\notin B} (\Pr[x = b] - \Pr[y = b]) \cdot \mathsf{val}_{\mathsf{min}} \\
&\qquad = |X - Y|_1(\mathsf{val}_{\mathsf{max}} - \mathsf{val}_{\mathsf{min}})/2 \leq |X - Y|_1 \cdot \mathsf{weight}_P/2.
\end{aligned}$$

□

Lemma 4. *For every regular evaluation program P of width d and length n,*

$$\mathsf{weight}_P \leq 2 \sum_{\{i,j\} \subset [d]} |q((n,i)) - q((n,j))|.$$

Proof. Consider the following game: $2d$ pebbles are placed on the real numbers $0 \leq q_1, \ldots, q_{2d} \leq 1$. At each step of the game one can choose two pebbles such that their distance is at least 2δ (for $\delta \geq 0$) and move each of them a distance of δ toward the other. The gain of that step is 2δ (that is, the total translation of the two pebbles in that step). The goal is to maximize the total gain that one can obtain in an unlimited number of steps, that is, the total translation of all pebbles in all steps.

Consider the game that starts with $2d$ pebbles placed on the real numbers

$$0 \leq q((n,1)), q((n,1)), q((n,2)), q((n,2)), \ldots, q((n,d)), q((n,d)) \leq 1.$$

By Equation (1), for every $t \in [n-1]$, one can start with 2 pebbles placed on each number $q((t+1,i))$ and end with 2 pebbles placed on each number $q((t,i))$, for $i \in [d]$, by applying d steps of the game described above (one step for each node in V_t). The total gain of these d steps is just the total weight of the edges in between V_t and V_{t+1}. Note that for this to hold we use regularity.

To complete the proof, we will show that if we start with pebbles placed at $q_1, \ldots, q_{2d}$, then the total possible gain in the pebble game is $L = \sum_{\{i,j\}} |q_i - q_j|$.

Without loss of generality, we can assume that each step operates on two adjacent pebbles. This is true because if in a certain step pebbles a, b are moved a distance of δ toward each other, and there is a pebble c in between a and b, one could reach the same final position (i.e., the same final position of all pebbles after that step), but with a higher gain, by first moving a and c a distance of δ' toward each other (for a small enough δ'), and then b and c a distance of δ' toward each other and finally a and b a distance of $\delta - \delta'$ toward each other.

If a step operates on two adjacent pebbles a, b, then for any other pebble c the sum of the distance between a and c and the distance between b and c remains the same (since c is not between a and b), while the distance between a and b decreases by 2δ (where 2δ is the gain of the step). Altogether, L decreases by exactly 2δ, the gain of the step. Since L cannot be negative, the total gain in the pebble game is bounded by the initial L. Since we can decrease L to be arbitrarily close to 0 (by operating on adjacent pebbles), the bound on the possible gain in the pebble game is tight.

□

4 The Generator

Our pseudorandom generator is a variant of the space generator of Impagliazzo, Nisan and Wigderson [INW94] (with different parameters). We think of this generator as a binary tree of extractors, where at each node of the tree an extractor is applied on the random bits used by the sub-tree rooted at the left-hand child of the node to obtain "recycled" random bits that are used by the sub-tree rooted at the right-hand child of the node (see for example [RR99]). We present our generator recursively, using extractors. We use the extractors constructed by Goldreich and Wigderson [GW97], using random walks on expander graphs and the expander mixing lemma.

The GW Extractor

Fix two integers n and d. Assume, for simplicity, that n is a power of 2. Let $\epsilon > 0$ be an error parameter that we are aiming for. Let

$$\beta = \frac{\epsilon}{2d^2 \log n},$$

and note that $\log(1/\beta) = O(\log d + \log\log n + \log(1/\epsilon))$. Let $k = \Theta(\log(1/\beta))$ be an integer, to be determined below.

For every $1 \leq i \leq \log n$, let

$$E_i : \{0,1\}^{ki} \times \{0,1\}^k \longrightarrow \{0,1\}^{ki}$$

be an (extractor) function such that the following holds: If $z_0, \ldots, z_i \sim U_k$ (and are independent), then for any event A depending only on $z = (z_0, \ldots, z_{i-1})$ such that $\Pr_z(A) \geq \beta$, the distribution of $E_i(z|A, z_i)$ is β-close to the uniform distribution. Explicit constructions of such functions were given in [GW97]. Fix $k = \Theta(\log(1/\beta))$ to be the length needed in their construction.

The Pseudorandom Generator

For $0 \leq i \leq \log n$, define a (pseudorandom generator) function

$$G_i : \{0,1\}^{k(i+1)} \longrightarrow \{0,1\}^{2^i}$$

recursively as follows. Let $y_0, \ldots, y_{\log n} \in \{0,1\}^k$. For $i = 0$, define $G_0(y_0)$ to be the first bit of y_0 (we use only the first bit of y_0, for simplicity of notation). For $1 \leq i \leq \log n$, define

$$G_i(y_0, \ldots, y_i) = G_{i-1}(y_0, \ldots, y_{i-1}) \circ G_{i-1}(E_i((y_0, \ldots, y_{i-1}), y_i)).$$

That is, G_i is generated in three steps: (1) generate 2^{i-1} bits by applying G_{i-1} on $(y_0, \ldots, y_{i-1})$; (2) apply the extractor E_i with seed y_i on $(y_0, \ldots, y_{i-1})$ to obtain $(y'_0, \ldots, y'_{i-1}) \in \{0,1\}^{ki}$; and (3) generate 2^{i-1} additional bits by applying G_{i-1} on $(y'_0, \ldots, y'_{i-1})$.

Our generator is

$$G = G_{\log n} : \{0,1\}^{k(\log n+1)} \longrightarrow \{0,1\}^n.$$

Analysis

The following theorem shows that G works.

Theorem 5. *For every evaluation program P (not necessarily regular) of width d and length 2^i+1,*

$$\left| \mathop{\mathbb{E}}_{y \sim U_{k(i+1)}} [\mathsf{val}_P(G_i(y))] - \mathop{\mathbb{E}}_{u \sim U_{2^i}} [\mathsf{val}_P(u)] \right| \leq i \cdot (d+1) \cdot \beta \cdot \mathsf{weight}_P.$$

Proof. We prove the statement by induction on i. For $i = 0$, the statement is trivially true, since $G_0(y)$ is uniformly distributed. To prove the statement for larger i, fix an evaluation program P that reads 2^i bits. We write $\mathsf{weight}_P = \mathsf{weight}_Q + \mathsf{weight}_R$, where weight_Q is the weight of edges in

the first half of the program, and weight_R is the weight of edges in the second half. Let $z \sim U_{ki}$, $y_i \sim U_k$ and $u_1, u_2 \sim U_{2^{i-1}}$. We need to bound,

$$\begin{aligned}&\left|\,\mathbb{E}\left[\mathsf{val}_P(G_i(z,y_i))\right] - \mathbb{E}\left[\mathsf{val}_P(u_1,u_2)\right]\,\right| \\ &\quad\le \left|\,\mathbb{E}\left[\mathsf{val}_P(G_{i-1}(z),u_2)\right] - \mathbb{E}\left[\mathsf{val}_P(u_1,u_2)\right]\,\right| + \left|\,\mathbb{E}\left[\mathsf{val}_P(G_i(z,y_i))\right] - \mathbb{E}\left[\mathsf{val}_P(G_{i-1}(z),u_2)\right]\,\right|. \quad (2)\end{aligned}$$

By Proposition 2, the first term is equal to $|\mathbb{E}\left[\mathsf{val}_P(G_{i-1}(z))\right] - \mathbb{E}\left[\mathsf{val}_P(u_1)\right]|$, which is at most $(i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_Q$ by the inductive hypothesis.

The second term equals

$$\left|\mathop{\mathbb{E}}_{z}\left[\mathop{\mathbb{E}}_{y_i}\left[\mathsf{val}_P(G_{i-1}(z),G_{i-1}(E_i(z,y_i)))\right] - \mathop{\mathbb{E}}_{u_2}\left[\mathsf{val}_P(G_{i-1}(z),u_2)\right]\right]\right|. \quad (3)$$

We shall bound (3) separately, depending on which of the vertices in the middle layer is reached by the program. Define the events $A_1,\ldots,A_d$, with $A_j = \{z : \mathrm{path}(G_{i-1}(z))_{2^{i-1}+1} = (2^{i-1}+1, j)\}$. Equation (3) is bounded from above by

$$\sum_{j=1}^{d}\Pr[A_j]\left|\mathop{\mathbb{E}}_{z|A_j}\left[\mathop{\mathbb{E}}_{y_i}\left[\mathsf{val}_P(G_{i-1}(z|A_j),G_{i-1}(E_i(z|A_j,y_i)))\right] - \mathop{\mathbb{E}}_{u_2}\left[\mathsf{val}_P(G_{i-1}(z|A_j),u_2)\right]\right]\right|.$$

Denote by R_j the evaluation program whose start vertex is $(2^{i-1}+1, j)$. Observe that if $z \in A_j$, then $\mathsf{val}_P(G_{i-1}(z),x) = \mathsf{val}_{R_j}(x)$. Thus,

$$(3) \le \sum_{j=1}^{d}\Pr[A_j]\left|\mathop{\mathbb{E}}_{z|A_j}\left[\mathop{\mathbb{E}}_{y_i}\left[\mathsf{val}_{R_j}(G_{i-1}(E_i(z|A_j,y_i)))\right] - \mathop{\mathbb{E}}_{u_2}\left[\mathsf{val}_{R_j}(u_2)\right]\right]\right|$$

Now if $\Pr[A_j] \le \beta$, the j'th term contributes at most

$$\beta\cdot\mathsf{weight}_R,$$

by Proposition 3. On the other hand, if $\Pr[A_j] \ge \beta$, then $E_i(z|A_j,y_i)$ is β-close to a uniformly random string. By Proposition 3 and the induction hypothesis, in this case the j'th term contributes at most

$$\Pr[A_j]\left((i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_R + \beta\cdot\mathsf{weight}_R/2\right).$$

Therefore,

$$\begin{aligned}(3) &\le d\cdot\beta\cdot\mathsf{weight}_R + \sum_{j=1}^{d}\Pr[A_j]\left((i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_R + \beta\cdot\mathsf{weight}_R\right) \\ &\le (i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_R + (d+1)\cdot\beta\cdot\mathsf{weight}_R.\end{aligned}$$

Putting the bounds for the two terms in (2) together, we get

$$\begin{aligned}(2) &\le (i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_Q + (i-1)\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_R + (d+1)\cdot\beta\cdot\mathsf{weight}_R \\ &= (i-1)\cdot(d+1)\cdot\beta\cdot(\mathsf{weight}_Q + \mathsf{weight}_R) + (d+1)\cdot\beta\cdot\mathsf{weight}_R \\ &\le i\cdot(d+1)\cdot\beta\cdot\mathsf{weight}_P,\end{aligned}$$

as required. □

Finally, we prove the main theorem of the paper.

Theorem 6. *There is an efficiently computable function $G : \{0,1\}^s \to \{0,1\}^n$ with*

$$s = O((\log d + \log\log n + \log(1/\epsilon))\log n),$$

such that if $u \sim U_n, y \sim U_s$ and B is any regular branching program of length $n+1$ and width d,

$$\left|\Pr[B(G(y)) = 1] - \Pr[B(u) = 1]\right| \leq \epsilon.$$

Proof. Set $G = G_{\log n}$ as in the construction above. The seed length to the generator is bounded by $O(k \log n) = O((\log d + \log\log n + \log(1/\epsilon))\log n)$ as required.

Given a branching program B, we make it an evaluation program P, by labeling every vertex v by the probability of reaching the accept vertex with a uniform random walk starting at v. We thus see that for any n bit string x, $B(x) = \mathsf{val}_P(x)$. From Theorem 5, it follows that

$$|\Pr[B(G(y)) = 1] - \Pr[B(u) = 1]| \leq (\log n) \cdot (d+1) \cdot \beta \cdot \mathsf{weight}_P.$$

By Lemma 4, $\mathsf{weight}_P \leq 2(d-1)$. Thus the error is at most $2d^2(\log n)\beta \leq \epsilon$, according to the choice of β. □

5 Biased Distributions Fool Small Width

Suppose we have a regular branching program B of length n and width d.

Let D be a distribution over $\{0,1\}^{n-1}$. For $\alpha \geq 0$, we say that D is α-biased (with respect to the branching program B) if for $x = (x_1, \ldots, x_{n-1}) \sim D$ the following holds: for every $t \in [n-1]$ and every $v \in V_t$ such that $\Pr_x[\mathrm{path}(x)_t = v] \geq \alpha$, the distribution of x_t conditioned on the event $\mathrm{path}(x)_t = v$ is α-close to uniform, that is, $|\Pr_x[x_t = 1 \mid \mathrm{path}(x)_t = v] - 1/2| \leq \alpha/2$.

The following theorem shows that the distribution of the node in the branching program reached by an α-biased random walk is $(\mathrm{poly}(d)\cdot\alpha)$-close to the distribution of the node reached by a uniform random walk.

Theorem 7. *Let P be a regular evaluation program of length n. Let $\alpha \geq 0$. Let D be an α-biased distribution (with respect to P). Then,*

$$\left|\mathop{\mathbb{E}}_{x\sim D}[\mathsf{val}_P(x)] - \mathop{\mathbb{E}}_{u\sim U_{n-1}}[\mathsf{val}_P(u)]\right| \leq \alpha \cdot \mathsf{weight}_P/2.$$

Before proving the theorem, we note that it can be shown by similar arguments that the distribution defined by G from the previous section is α-biased, with small α. Using the theorem, this also implies that G fools regular branching programs.

Proof. We prove the theorem using a hybrid argument. For each $t \in \{0, \ldots, n-1\}$, define the distribution D_t to be the same as D on the first t bits, and the same as U_{n-1} on the remaining bits. Thus $D_0 = U_{n-1}$ and $D_{n-1} = D$. By the triangle inequality, we have that

$$\left|\mathop{\mathbb{E}}_{x\sim D}[\mathsf{val}_P(x)] - \mathop{\mathbb{E}}_{u\sim U_{n-1}}[\mathsf{val}_P(u)]\right| \leq \sum_{t=0}^{n-2}\left|\mathop{\mathbb{E}}_{x\sim D_t}[\mathsf{val}_P(x)] - \mathop{\mathbb{E}}_{y\sim D_{t+1}}[\mathsf{val}_P(y)]\right|.$$

For $t \in \{1, \ldots, n-1\}$, let weight_t denote the weight of the edges going out of V_t. We claim that the t'th term in the sum is bounded by $\alpha \cdot \mathsf{weight}_{t+1}/2$. The sum of all terms is thus at most $\alpha/2 \cdot \sum_{t=1}^{n-1} \mathsf{weight}_t = \alpha \cdot \mathsf{weight}_P/2$, as required.

To bound the t'th term, let z be distributed according to the first $t+1$ bits of D. Let $z_{\leq t}$ denote the first t bits of z, and let z_{t+1} be the $t+1$'st bit of z. Let u_{t+1} denote a uniform bit. Since all bits in D_t, D_{t+1} after the $t+1$'st bit are uniform, Proposition 2 implies that the t'th term in the sum is equal to

$$\left| \mathop{\mathbb{E}}_{z_{\leq t}} \left[\mathop{\mathbb{E}}_{z_{t+1}} [\mathsf{val}_P(z_{\leq t}, z_{t+1})] - \mathop{\mathbb{E}}_{u_{t+1}} [\mathsf{val}_P(z_{\leq t}, u_{t+1})] \right] \right|.$$

For every vertex v in V_{t+1}, define the event A_v to be the event that $\mathrm{path}(z_{\leq t})_{t+1} = v$, and let R_v denote the evaluation program with two layers whose start vertex is v. R_v involves only two edges, since only the edges leading out of v are traversable. We have that $\sum_{v \in V_{t+1}} \mathsf{weight}_{R_v} = \mathsf{weight}_{t+1}$. Observe that if $z_{\leq t} \in A_v$, then $\mathsf{val}_P(z_{\leq t}, y) = \mathsf{val}_{R_v}(y)$. So we can bound the t'th term from above by

$$\sum_{v \in V_{t+1}} \left| \Pr[A_v] \left(\mathop{\mathbb{E}}_{z_{t+1}|A_v} [\mathsf{val}_{R_v}(z_{t+1}|A_v)] - \mathop{\mathbb{E}}_{u_{t+1}} [\mathsf{val}_{R_v}(u_{t+1})] \right) \right|. \tag{4}$$

There are two cases we need to consider. The first case is when v admits $\Pr[A_v] \geq \alpha$. In this case, $z_{t+1}|A_v$ is α-close to uniform, and by Proposition 3, the v'th term is bounded by $\Pr[A_v] \cdot \alpha \cdot \mathsf{weight}_{R_v}/2$. The second case is when v admits $\Pr[A_v] < \alpha$. In this case, Proposition 3 tells us that the v'th term is bounded by $\alpha \cdot \mathsf{weight}_{R_v}/2$. To conclude,

$$(4) \leq \alpha \cdot \sum_{v \in V_{t+1}} \mathsf{weight}_{R_v}/2 = \alpha \cdot \mathsf{weight}_{t+1}/2.$$

□

As a corollary, we get that α-biased distributions are pseudorandom for regular branching programs of bounded width.

Corollary 8. *Let B be a regular branching program of length n and width d. Let $\alpha \geq 0$. Let D be an α-biased distribution (with respect to B). Then,*

$$\left| \Pr_{x \sim D}[B(x) = 1] - \Pr_{u \sim U_{n-1}}[B(u) = 1] \right| \leq \alpha(d-1).$$

Proof. Let P be the evaluation program obtained by labeling every vertex of B with the probability of accepting a uniform input starting at that vertex. Since P is regular and has width d, $\mathsf{weight}_P \leq 2(d-1)$ by Lemma 4. Apply Theorem 7 to complete the proof. □

Remark 9. *Corollary 8 tells us that in order to fool regular constant width branching programs with constant error, we can use α-biased distributions, with α a small enough constant. This statement is false for non-regular programs, as we now explain. Consider the function tribes_n that is defined as follows: Let k be the maximal integer so that $(1-2^{-k})^n \leq 1/2$. The function tribes_n takes as input nk bits $x = (x_{i,j})_{i \in [n], j \in [k]}$ and $\mathrm{tribes}_n(x) = \bigvee_{i \in [n]} \bigwedge_{j \in [k]} x_{i,j}$. The tribes function has a natural width 3 branching program. This program is, however, not regular. Even a very strong notion of*

α-biased distribution does not fool it, as long as $\alpha \gg 1/\log n$. This is true as if all the bits in D are, say, $(10/\log n)$-biased towards 1 and all of them are independent*, then the expectation of the tribes function with respect to D is $\Omega(1)$-far from the expectation of the tribes function with respect to the uniform distribution.*

Acknowledgements

We would like to thank Zeev Dvir, Omer Reingold and David Zuckerman for helpful discussions.

References

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.

[AKS87] Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In *STOC*, pages 132–140. ACM, 1987.

[Bar89] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, February 1989.

[BNS89] László Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences (extended abstract). In *STOC*, pages 1–11. ACM, 1989.

[GW97] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Struct. Algorithms*, 11(4):315–343, 1997.

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

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis94] Noam Nisan. RL $\subseteq$ SC. *Computational Complexity*, 4:1–11, 1994.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, August 1993.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *STOC*, pages 159–168, 1999.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci*, 4(2):177–192, 1970.

[SZ] Michael Saks and David Zuckerman. Personal communication.

[SZ99] Michael E. Saks and Shiyu Zhou. $\mathrm{BP}_H\mathrm{space}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *J. Comput. Syst. Sci*, 58(2):376–403, 1999.

A The Bounded Probability Case

We now show that the generator fools a more general class of programs, namely, branching programs in which every vertex is hit with either zero or non-negligible probability by a truly random input. Such programs are not necessarily regular, but every regular program can be shown to have this property. We start by showing that the weight of such programs can be bounded.

Suppose P is an evaluation program of length n and width d. For every vertex v in the program, we denote by $p(v)$ the probability that the program reaches the vertex v starting at $(1,1)$, according to the uniform distribution. Recall that $q(v)$ is the value of the vertex v in the program P.

For technical reasons, we need the following definition. For a given evaluation program P of length n and width d, define P', the *non-redundant* part of P, as the program obtained by removing from P all vertices v with $p(v) = 0$. The non-redundant part of P is not necessarily an evaluation program, according to our definition, as some of its layers may have less than d vertices. Nevertheless, P' has a natural notion of weight induced by P, by assigning every vertex in P' the same value as its value in P. The program P' also has a natural structure of layers, induced by P: for $t \in [n]$, the vertices in V'_t are those vertices v in V_t so that $p(v) > 0$.

Lemma 10. *Let P be an evaluation program, and $\gamma > 0$ be such that for every vertex v in P, either $p(v) = 0$ or $p(v) \geq \gamma$. Then $\mathsf{weight}_{P'} \leq 2/\gamma^2$, where P' is the non-redundant part of P.*

Proof. The proof is a fractional version of the pebble argument used in the regular case. We play the following pebble game. We start with a number of pebbles, located at *positions* $q_1, q_2, \ldots, q_\ell \in [0,1]$. The pebbles also have corresponding *heights* $p_1, \ldots, p_\ell > 0$ that add up to 1: $\sum_{i=1}^{\ell} p_i = 1$. The rules of the game are as follows. In each step, we are allowed to pick a parameter $\eta > 0$ and two pebbles at positions a, b, each of which has height at least η. We then reduce the heights of each of these pebbles by η, and add a new pebble of height 2η at position $(a+b)/2$. The *gain* in this step is $\eta^2|a-b|$. If two pebbles are at the same position, then we treat them as a single pebble whose height is just the sum of the heights of the pebbles. The sum of heights of the pebbles is 1 throughout the game.

First, we observe that the program P' defines a way to achieve a gain of at least $(\gamma/2)^2 \cdot \mathsf{weight}_{P'}$, as follows. We do so in $n-1$ steps, indexed by $t \in \{n-1, n-2, \ldots, 1\}$. The way we start the game is *specified* by V'_n: for each i such that $p((n,i)) > 0$, *associate* the vertex (n,i) in P' with a pebble at position $q_i = q((n,i))$ and height $p((n,i))$. We maintain the following property throughout the game: before we start the t'th step, for every pebble at the current configuration of the game, the sum $\sum_w p(w)$, with w associated with the pebble, is the height of the pebble. Here is how we perform the t'th step. From the pebble configuration specified by V'_{t+1}, we obtain the configuration specified by V'_t, by applying $|V'_t|$ fractional pebble moves, as follows. In each one of these moves, we pick a vertex $v \in V'_t$, we choose $\eta = p(v)/2 > 0$, and we choose a to be the position of the pebble associated with $\mathrm{next}_0(v)$ and b to be the position of the pebble associated with $\mathrm{next}_1(v)$. We then apply the pebble move defined by η, a, b, and *associate* the vertex v with the pebble at position $(a+b)/2$. Since $p(v) \geq \gamma$, the gain of this step is at least $(\gamma/2)^2|a-b|$. The total gain obtained by reaching the configuration specified by V'_t from that specified by V'_{t+1} is thus at least $(\gamma/2)^2$ times

the weight of the layer. Continuing in this way for the whole program, we get a sequence of pebble moves with total gain at least $(\gamma/2)^2 \cdot \mathsf{weight}_{P'}$.

Next, we show that the total gain in any game with any starting configuration is at most $1/2$ (again, this bound holds even for an unbounded number of moves). For any configuration of ℓ pebbles at positions $q_1, \ldots, q_\ell$ and heights $p_1, \ldots, p_\ell$, define the quantity $L = \sum_{\{i,j\} \subset [\ell]} p_i p_j |q_i - q_j|$. We claim that in any valid fractional pebble move that is defined by η, a, b, this quantity must decrease by at least $\eta^2 |a - b|$. To see this, observe that if $c \notin \{a, b\}$ is a position of a pebble, then the sum of terms involving c in L can only decrease: if p_a, p_b, p_c are the heights of the pebbles at positions a, b, c,

$$p_c p_a |a - c| + p_c p_b |b - c| \geq p_c (p_a - \eta)|a - c| + p_c (p_b - \eta)|b - c| + p_c 2\eta |(a + b)/2 - c|,$$

as

$$(|a - c| + |b - c|)/2 \geq |(a + b)/2 - c|,$$

by convexity. Moreover, the pebbles at positions a, b reduce the sum by

$$p_a p_b |a - b| - \big((p_a - \eta)(p_b - \eta)|a - b| + (p_a - \eta)2\eta |a - b|/2 + (p_b - \eta)2\eta |a - b|/2\big) = |a - b|\eta^2.$$

Since L is always non-negative, the initial L, which is

$$L_{\text{initial}} = \sum_{\{i,j\} \subset [k]} p_i p_j |q_i - q_j| \leq \sum_{\{i,j\} \subset [k]} p_i p_j < 1/2$$

for some $k \in \mathbb{N}$, is thus an upper bound on the total gain possible in the fractional pebble game.

To conclude,

$$(\gamma/2)^2 \cdot \mathsf{weight}_{P'} \leq \text{ total gain of game } < 1/2.$$

□

Lemma 10 and Theorem 5 imply that the pseudorandom generator defined earlier fools branching programs that do not have low probability vertices.

Theorem 11. *Let B be a branching program, and $\gamma > 0$ be such that for every vertex v in the program, either $p(v) = 0$ or $p(v) \geq \gamma$. Let $G = G_{\log n}$ be the generator as defined above. Then if y, u are distributed uniformly at random (as in Theorem 6),*

$$\big|\Pr[B(G(y)) = 1] - \Pr[B(u) = 1]\big| \leq 2\epsilon/\gamma^2.$$

Proof. We define the evaluation program P by setting $q(v)$ to be the probability of accepting a uniform input starting at the vertex v. Let P' be the non-redundant part of P. By Lemma 10, $\mathsf{weight}_{P'} \leq 2/\gamma^2$. In terms of functionality, P and P' are equivalent. The proof of Theorem 5 thus tells us that the error of the generator is at most $\beta(\log n)(d + 1)\mathsf{weight}_{P'} \leq 2\epsilon/\gamma^2$, by the choice of β. □

It follows from Theorem 11 that one can efficiently construct a generator that ϵ-fools branching programs in which every vertex is reached with probability either zero or at least γ, using a seed of length $O((\log \log n + \log(1/\epsilon) + \log(1/\gamma)) \log n)$, as we can assume $d \leq O(1/\gamma)$.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de