Pseudorandomness and Fourier Growth Bounds for Width-3 Branching Programs

Thomas Steinke*[1], Salil Vadhan†[1], and Andrew Wan‡[2]

1 **School of Engineering and Applied Sciences, Harvard University**
33 Oxford Street, Cambridge MA
{tsteinke,salil}@seas.harvard.edu

2 **Simons Institute for the Theory of Computing, UC Berkeley**
atw12@seas.harvard.edu

Abstract

We present an explicit pseudorandom generator for oblivious, read-once, width-3 branching programs, which can read their input bits in any order. The generator has seed length $\tilde{O}(\log^3 n)$. The previously best known seed length for this model is $n^{1/2+o(1)}$ due to Impagliazzo, Meka, and Zuckerman (FOCS '12). Our work generalizes a recent result of Reingold, Steinke, and Vadhan (RANDOM '13) for *permutation* branching programs. The main technical novelty underlying our generator is a new bound on the Fourier growth of width-3, oblivious, read-once branching programs. Specifically, we show that for any $f : \{0,1\}^n \to \{0,1\}$ computed by such a branching program, and $k \in [n]$,

$$\sum_{s \subseteq [n] : |s| = k} \left| \widehat{f}[s] \right| \leq n^2 \cdot (O(\log n))^k,$$

where $\widehat{f}[s] = \mathop{\mathbb{E}}_{U} \left[f[U] \cdot (-1)^{s \cdot U} \right]$ is the standard Fourier transform over $\mathbb{Z}_2^n$. The base $O(\log n)$ of the Fourier growth above is tight up to a factor of $\log \log n$.

1998 ACM Subject Classification F.1 Computation by Abstract Devices

Keywords and phrases Pseudorandomness, Branching Programs, Discrete Fourier Analysis

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2014.885

1 Introduction

1.1 Pseudorandom Generators for Space-Bounded Computation

A major open problem in the theory of pseudorandomness is to construct an "optimal" pseudorandom generator for space-bounded computation. That is, we want an explicit algorithm that stretches a uniformly random seed of length $O(\log n)$ to n bits that cannot be distinguished from uniform by any $O(\log n)$-space algorithm (which receives the pseudorandom bits one at a time, in a streaming fashion, and may be nonuniform). Such a generator would imply that every randomized algorithm can be derandomized with only a constant-factor increase in space (RL = L), and would also have a variety of other applications, such as in streaming algorithms [24], deterministic dimension reduction and SDP rounding [35, 15],

* Supported by NSF grant CCF-1116616 and the Lord Rutherford Memorial Research Fellowship.
† Supported in part by NSF grant CCF-1116616, US-Israel BSF grant 2010196, and a Simons Investigator Award.
‡ Part of this work was completed while at Harvard University and supported by NSF grant CCF-0964401.

17th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'14) / 18th Int'l Workshop on Randomization and Computation (RANDOM'14).
Editors: Klaus Jansen, José Rolim, Nikhil Devanur, and Cristopher Moore; pp. 885–899

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

hashing [12], hardness amplification [21], almost k-wise independent permutations [25], and cryptographic pseudorandom generator constructions [20].

To construct a pseudorandom generator for space-bounded algorithms using space s, it suffices to construct a generator that is pseudorandom against ordered branching programs of width 2^s. A branching program[1] B is a non-uniform model of space-bounded computation that reads one input bit at a time, maintaining a state in $[w] = \{1, \ldots, w\}$, where w is called the width of B. At each time step $i = 1, \ldots, n$, B can read a different input bit $x_{\pi(i)}$ (for some permutation π) and uses a different state transition function $T_i : [w] \times \{0,1\} \to [w]$. It is often useful to think of a branching program as a directed acyclic graph consisting of $n+1$ layers of w vertices each, where the i^{th} layer corresponds to the state at time i. The transition function defines a bipartite graph between consecutive layers, where we connect state s in layer $i-1$ to states $T_i(s,0)$ and $T_i(s,1)$ in layer i (labeling those edges 0 and 1, respectively). Most previous constructions of pseudorandom generators for space-bounded computations consider *ordered* branching programs, where the input bits are read in order – that is, $\pi(i) = i$.

The classic work of Nisan [30] gave a generator with seed length $O(\log^2 n)$ that is pseudorandom against ordered branching programs of polynomial width. Despite intensive study, this is the best known seed length for ordered branching programs even of width 3, but a variety of works have shown improvements for restricted classes such as branching programs of width 2 [33, 4], and regular or permutation branching programs (of constant width) [8, 9, 26, 13, 37]. For width 3, hitting set generators (a relaxation of pseudorandom generators) have been constructed [39, 17]. The vast majority of these works are based on Nisan's original generator or its variants by Impagliazzo, Nisan, and Wigderson [23] and Nisan and Zuckerman [31], and adhere to a paradigm that seems unlikely to yield generators against general logspace computations with seed length better than $\log^{1.99} n$ (see [9]).

All known analyses of Nisan's generator and its variants rely on the order in which the output bits are fed to the branching program (given by the permutation π). The search for new ideas leads us to ask: Can we construct a pseudorandom generator whose analysis does not depend on the order in which the bits are read? A recent line of work [5, 22, 32] has constructed pseudorandom generators for unordered branching programs (where the bits are fed to the branching program in an arbitrary, fixed order); however, none match both the seed length and generality of Nisan's result. For unordered branching programs of length n and width w, Impagliazzo, Meka, and Zuckerman [22] give seed length $O((nw)^{1/2+o(1)})$ improving on the linear seed length $(1-\Omega(1)) \cdot n$ of Bogdanov, Papakonstantinou, and Wan [5].[2] Reingold, Steinke, and Vadhan [32] achieve seed length $O(w^2 \log^2 n)$ for the restricted class of *permutation* branching programs, in which $T_i(\cdot, b)$ is a permutation on $[w]$ for all $i \in [n]$ and $b \in \{0,1\}$.

Recently, a new approach for constructing pseudorandom generators has been suggested in the work of Gopalan et al. [17]; they constructed pseudorandom generators for read-once CNF formulas and combinatorial rectangles, and hitting set generators for width-3 branching programs, all having seed length $\tilde{O}(\log n)$ (even for polynomially small error). Their basic generator (e.g. for read-once CNF formulas) works by pseudorandomly partitioning the bits into several groups and assigning the bits in each group using a small-bias generator [29]. A

[1] In this work and the definition we give here, we consider read-once, oblivious branching programs, and refer to them simply as branching programs for brevity.

[2] A generator with seed length $\tilde{O}(\sqrt{n} \log w)$ is given in [32]. The generator in [22] also extends to branching programs that read their inputs more than once and in an adaptively chosen order, which is more general than the model we consider.

key insight in their analysis is that the small-bias generator only needs to fool the function "on average," where the average is taken over the possible assignments to subsequent groups, which is a weaker requirement than fooling the original function or even a random restriction of the original function. (For a more precise explanation, see Section 4.)

The analysis of Gopalan et al. [17] does not rely on the order in which the output bits are read, and the previously mentioned work by Reingold, Steinke, and Vadhan [32] uses Fourier analysis of branching programs to show that the generator of Gopalan et al. fools unordered permutation branching programs. In this work we further develop Fourier analysis of branching programs and show that the pseudorandom generator of Gopalan et al. with seed length $\tilde{O}(\log^3 n)$ fools width-3 branching programs:

▶ **Theorem 1** (Main Result). *There is an explicit pseudorandom generator* $G : \{0,1\}^{O(\log^3 n \cdot \log\log n)} \to \{0,1\}^n$ *fooling oblivious, read-once (but unordered), branching programs of width* 3 *and length* n.

The previous best seed length for this model is the aforementioned length of $O(n^{1/2+o(1)})$ given in [22]. The construction of the generator in Theorem 1 is essentially the same as the generator of Gopalan et al. [17] for read-once CNF formulas, which was used by Reingold et al. [32] for permutation branching programs. In our analysis, we give a new bound on the Fourier mass of width-3 branching programs.

1.2 Fourier Growth of Branching Programs

For a function $f : \{0,1\}^n \to \mathbb{R}$, let $\widehat{f}[s] = \mathop{\mathbb{E}}_{U}\left[f[U] \cdot (-1)^{s \cdot U}\right]$ be the standard Fourier transform over $\mathbb{Z}_2^n$, where U is a random variable distributed uniformly over $\{0,1\}^n$ and $s \subseteq [n]$ or, equivalently, $s \in \{0,1\}^n$. The Fourier mass of f (also called the spectral norm of f), defined as $L(f) := \sum_{s \neq \emptyset} |\hat{f}[s]|$, is a fundamental measure of complexity for Boolean functions (e.g., see [18]), and its study has applications to learning theory [27, 28], communication complexity [19, 1, 38, 34], and circuit complexity [7, 10, 11]. In the study of pseudorandomness, it is well-known that small-bias generators[3] with bias ε/L (which can be sampled using a seed of length $O(\log(n \cdot L/\varepsilon))$ [29, 2]) will ε-fool any function whose Fourier mass is at most L. Width-2 branching programs have Fourier mass at most $O(n)$ [4, 33] and are thus fooled by small-bias generators with bias ε/n. Unfortunately, such a bound does not hold even for very simple width-3 programs. For example, the 'mod 3 function,' which indicates when the hamming weight of its input is a multiple of 3 has Fourier mass exponential in n.

However, a more refined measure of Fourier mass is possible and often useful: Let $L^k(f) = \sum_{|s|=k} |\hat{f}[s]|$ be the level-k Fourier mass of f. A bound on the Fourier growth of f, or the rate at which $L^k(f)$ grows with k, was used by Mansour [28] to obtain an improved query algorithm for polynomial-size DNF; the junta approximation results of Friedgut [16] and Bourgain [6] are proven using approximating functions that have slow Fourier growth. This notion turns out to be useful in the analysis of pseudorandom generators as well: Reingold et al. [32] show that the generator of Gopalan et al. [17] will work if there is a good bound on the Fourier mass of low-order coefficients. More precisely, they show that for any class $\mathcal{C}$ of functions computed by branching programs that is closed under restrictions and decompositions and satisfies $L^k(f) \leq \mathrm{poly}(n) \cdot c^k$ for every k and $f \in \mathcal{C}$, there is a

[3] A small-bias generator with bias μ outputs a random variable $X \in \{0,1\}^n$ such that $\left|\mathop{\mathbb{E}}_{X}\left[(-1)^{s \cdot X}\right]\right| \leq \mu$ for every $s \subset [n]$ with $s \neq \emptyset$.

pseudorandom generator with seed length $\tilde{O}(c \cdot \log^2 n)$ that fools every $f \in \mathcal{C}$. They then bound the Fourier growth of permutation branching programs (and the even more general model of "regular" branching programs, where each layer is a regular bipartite graph) to obtain a pseudorandom generator for permutation branching programs:

▶ **Theorem 2** ([32, Theorem 1.4]). *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a length-n, width-w, read-once, oblivious,* regular *branching program. Then, for all $k \in [n]$, $L^k(f) \leq (2w^2)^k$.*

In particular, the mod 3 function over $O(k)$ bits, which is computed by a permutation branching program of width 3, has Fourier mass $2^{\Theta(k)}$ a level k. However, the Tribes function,[4] which is also computed by a width-3 branching program, has Fourier mass $\Omega(\log n / \log k)^k$ at level k, so the bound in Theorem 2 does not hold for non-regular branching programs even of width 3.

The Coin Theorem of Brody and Verbin [9] implies a related result: essentially, a function computed by a width-w, length-n branching program cannot distinguish product distributions on $\{0,1\}^n$ any better than a function satisfying $L^k(f) \leq (\log n)^{O(wk)}$ for all k. To be more precise, if $X \in \{0,1\}^n$ is n independent samples from a coin with bias β (that is, each bit has expectation $(1+\beta)/2$), then $\mathop{\mathbb{E}}_X [f[X]] = \sum_s \widehat{f}[s]\beta^{|s|}$. If $L^k(f) \leq (\log n)^{O(wk)}$ for all k, then

$$\left| \mathop{\mathbb{E}}_X [f[X]] - \mathop{\mathbb{E}}_U [f[U]] \right| = \left| \sum_{s \neq 0} \widehat{f}[s]\beta^{|s|} \right| \leq \sum_{k \in [n]} L^k(f)|\beta|^{|s|} \leq O(|\beta|(\log n)^{O(w)}),$$

assuming $|\beta| \leq 1/(\log n)^{O(w)}$. Brody and Verbin prove that, if f is computed by a length-n, width-w branching program, then $|\mathop{\mathbb{E}}_X [f[X]] - \mathop{\mathbb{E}}_U [f[U]]| \leq O(|\beta|(\log n)^{O(w)})$. Since distinguishing product distributions captures much of the power of a class of functions, this leads to the following conjecture.

▶ **Conjecture 3** ([32, Conjecture 8.1]). *For every constant w, the following holds. Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-w, read-once, oblivious branching program. Then*

$$L^k(f) \leq n^{O(1)} \cdot (\log n)^{O(k)},$$

where the constants in the $O(\cdot)$s may depend on w.

In this work, we prove this conjecture for $w = 3$:

▶ **Theorem 4** (Fourier Growth of Width 3). *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-3, read-once, oblivious branching program. Then, for all $k \in [n]$,*

$$L^k(f) := \sum_{s : |s| = k} |\widehat{f}[s]| \leq n^2 \cdot (O(\log n))^k .$$

This bound is the main contribution of our work and, when combined with the techniques of Reingold et al. [32], implies our main result (Theorem 1).

The Tribes function of [3] shows that the base of $O(\log n)$ of the Fourier growth in Theorem 4 is tight up to a factor of $\log \log n$. (See the full version of this paper for a proof.)

We also prove Conjecture 3 with $k = 1$ for any constant width w:

[4] The Tribes function (introduced by Ben-Or and Linial [3]) is a DNF formula where all the terms are the same size and every input appears exactly once. The size of the clauses in this case is chosen to give an asymptotically constant acceptance probability on uniform input.

▶ **Theorem 5.** *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-w, length-n, read-once, oblivious branching program. Then*

$$L^1(f) = \sum_{i \in [n]} |\widehat{f}[\{i\}] \leq (O(\log n))^{w-2}.$$

1.3 Techniques

The intuition behind our approach begins with two extreme cases of width-3 branching programs: permutation branching programs and branching programs in which every layer is a non-permutation layer. Permutation branching programs "mix" well: on a uniform random input, the distribution over states gets closer to uniform (in ℓ_2 distance) in each layer. We can use this fact with an inductive argument to achieve a bound of $2^{O(k)}$ on the level-k Fourier mass (this is the bound of Theorem 2).

For branching programs in which *every* layer is a non-permution layer, we can make use of an argument from the work of Brody and Verbin [9]: when we apply a random restriction (where each variable is kept free with probability roughly $1/k$) to such a branching program, the resulting program is 'simple' in that the width has collapsed to 2 in many of the remaining layers. This allows us to use arguments tailored to width-2 branching programs, which are well-understood. In particular, we can use the same concept of mixing as used for permutation branching programs.

To handle general width-3 branching programs, which may contain an arbitrary mix of permutation and non-permutation layers, we group the layers into "chunks" containing exactly one non-permutation layer each. Instead of using an ordinary random restriction, we consider a series of restrictions similar to those in Steinberger's "interwoven hybrids" technique [36] (in our argument each chunk will correspond to a single layer in [36]). In Section 3.1, we use such restrictions to show that the level-k Fourier mass of an arbitrary width-3 program can be bounded in terms of the level-k Fourier mass of a program D which has the following "pseudomixing" form: D can be split into $r \in [n]$ branching programs $D_1 \circ D_2 \circ \cdots \circ D_r$, where each D_i has at most $3k$ non-regular layers and the layer splitting consecutive D_is has width 2.

We then generalize the arguments used for width-2 branching programs to "pseudomixing" branching programs. We can show that each chunk D_i is either mixing or has small Fourier growth, which suffices to bound the Fourier growth of D.

2 Preliminaries

2.1 Branching Programs

We view a length-n, width-w **branching program** as a function $B : \{0,1\}^n \times [w] \to [w]$, which takes a start state $u \in [w]$ and an input string $x \in \{0,1\}^n$ and outputs a final state $B[x](u)$. We can view B as computing a Boolean function by fixing a start state and set of accept states in $[w]$. In this work we consider branching programs with random (or pseudorandom) inputs, in which case a program can be viewed as a Markov chain randomly taking initial states to final states. That is, B can be viewed as a matrix-valued function $B : \{0,1\}^n \to \{0,1\}^{w \times w}$ where $B[x]_{u,v} = 1$ if and only if $B[x](u) = v$. For a random variable X on $\{0,1\}^n$, we have $\mathop{\mathbb{E}}_X [B[X]] \in [0,1]^{w \times w}$, where the entry in the u^{th} row and v^{th} column $\mathop{\mathbb{E}}_X [B[X]]_{u,v}$ is the probability that B takes the initial state u to the final state v when given a random input from the distribution X. The matrix $\mathop{\mathbb{E}}_X [B[X]]$ is **stochastic**, that is, its

rows give probability distributions (i.e., they are non-negative and sum to 1). A **regular program** B has the property that the uniform distribution is a stationary distribution of the Markov chain $\mathop{\mathbb{E}}_{U}[B[U]]$, whereas, if B is a **permutation program**, the uniform distribution is stationary for $\mathop{\mathbb{E}}_{X}[B[X]]$ for *any* distribution X.

We write $B_1 \circ B_2$ to denote the *concatenation* of two branching programs, where the start state of B_2 is determined by the final state of B_1 on the input. Thus the matrix representation of $B_1 \circ B_2[x_1 \circ x_2]$ is given by $B_1[x_1] \cdot B_2[x_2]$. A length-n, width-w, **ordered branching program** (abbreviated **OBP**) is a program B that can be written $B = B_1 \circ B_2 \circ \cdots \circ B_n$, where each B_i is a length-1 width-w program. We refer to B_i as the i^{th} **layer** of B. We denote the **subprogram** of B from layer i to layer j by $B_{i\cdots j} := B_i \circ B_{i+1} \circ \cdots \circ B_j$. We sometimes consider branching programs of varying width – some layers have more vertices than others. The overall width of the program is the maximum width of any layer. This means that the edge layers B_i may give non-square matrices. For $i \in [n]$, if $B_i[x] \in \{0,1\}^{w \times w'}$, then we refer to w as the width of layer $i-1$ and w' as the width of layer i.

2.2 Fourier Analysis

Let $B : \{0,1\}^n \to \mathbb{R}^{w \times w'}$ be a matrix-valued function (such as given by a length-n, width-w branching program). Then we define the **Fourier transform** of B as a matrix-valued function $\widehat{B} : \{0,1\}^n \to \mathbb{R}^{w \times w'}$ given by

$$\widehat{B}[s] := \mathop{\mathbb{E}}_{U}\left[B[U]\chi_s(U)\right],$$

where $s \in \{0,1\}^n$ (or, equivalently, $s \subset [n]$) and

$$\chi_s(x) = (-1)^{\sum_i x_i \cdot s_i} = \prod_{i \in s}(-1)^{x(i)}.$$

We refer to $\widehat{B}[s]$ as the s^{th} **Fourier coefficient** of B, which has **order** (or degree) $|s|$. Note that this is equivalent to taking the real-valued Fourier transform of each of the $w \cdot w'$ entries of $B[x]$ separately, but we see in the following lemma that this matrix-valued Fourier transform is nicely compatible with matrix algebra.

▶ **Lemma 6.** *Let $A : \{0,1\}^n \to \mathbb{R}^{w \times w'}$ and $B : \{0,1\}^n \to \mathbb{R}^{w' \times w''}$ be matrix valued functions. Let X, Y, and U be independent random variables over $\{0,1\}^n$, where U is uniform. Let $s, t \in \{0,1\}^n$. Then we have the following.*

- *Decomposition: If $C[x \circ y] = A[x] \cdot B[y]$ for all $x, y \in \{0,1\}^n$, then $\widehat{C}[s \circ t] = \widehat{A}[s] \cdot \widehat{B}[t]$.*
- *Fourier Inversion for Matrices: $B[x] = \sum_s \widehat{B}[s]\chi_s(x)$.*
- *Parseval's Identity: $\sum_{s \in \{0,1\}^n} \left|\left|\widehat{B}[s]\right|\right|_{Fr}^2 = \mathop{\mathbb{E}}_{U}\left[||B[U]||_{Fr}^2\right]$.*

The Decomposition property is what makes the matrix-valued Fourier transform more convenient than separately taking the Fourier transform of the matrix entries as done by Bogdanov et al. [5]. If B is a length-n, width-w, ordered branching program, then, for all $s \in \{0,1\}^n$,

$$\widehat{B}[s] = \widehat{B}_1[s_1] \cdot \widehat{B}_2[s_2] \cdot \cdots \cdot \widehat{B}_n[s_n].$$

The **Fourier mass** of a matrix-valued function B is $L(B) := \sum_{s \neq 0} \left|\left|\widehat{B}[s]\right|\right|_2$, and the Fourier mass at level-k is $L^k(B) := \sum_{s \in \{0,1\}^n : |s| = k} \left|\left|\widehat{B}[s]\right|\right|_2$. We define $L^{\geq k}(B) := \sum_{k' \geq k} L^{k'}(B)$ and $L^{\leq k}(B)$, $L^{>k}(B)$, $L^{<k}(B)$ are defined analogously. The Fourier mass is unaffected by order:

▶ **Lemma 7.** *Let $B, B' : \{0,1\}^n \to \mathbb{R}^{w \times w}$ be matrix-valued functions satisfying $B[x] = B'[\pi(x)]$, where $\pi : [n] \to [n]$ is a permutation. Then, for all $s \in \{0,1\}^n$, $\widehat{B}[s] = \widehat{B'}[\pi(s)]$. In particular, $L(B) = L(B')$ and $L^k(B) = L^k(B')$ for all k.*

Lemma 7 implies that the Fourier mass of any read-once, oblivious branching program is equal to the Fourier mass of the corresponding ordered branching program.

3 Fourier Analysis of Width-3 Branching Programs

In this section we sketch the proof of our bound on the low-order Fourier mass of width-3, read-once, oblivious branching programs (Theorem 4). This is key to the analysis of our pseudorandom generator. See the full version of our paper (which is available online) for a complete proof.

To prove Theorem 4 we will consider the matrix valued function B of the branching program computing f. Note that $|\hat{f}[s]| \leq ||\widehat{B}[s]||_2$ for all s so $L^k(f) \leq L^k(B)$. We may also assume without loss of generality that the first and last layers of the program have width 2 (there is only one start state, and there are at most 2 accept states otherwise the program is trivial). The proof proceeds in two parts. The first part reduces the problem to one about branching programs of a special form, namely ones where many layers have been reduced to width-2. The second part uses the mixing properties of width-2 programs to bound the Fourier mass.

3.1 Part 1 – Reduction of Width by Random Restriction

First some definitions:

For $g \subset [n]$ and $x \in \{0,1\}^n$, define the **restriction of B to g using x – denoted** $B|_{\overline{g} \leftarrow x}$ – to be the branching program obtained by setting the inputs (layers of edges) of B outside g to values from x and leaving the inputs in g free. More formally,

$$B|_{\overline{g} \leftarrow x}[y] = B[\mathrm{Select}(g, y, x)], \quad \text{where} \quad \mathrm{Select}(g, y, x)_i = \left\{ \begin{array}{ll} y_i & i \in g \\ x_i & i \notin g \end{array} \right\}.$$

Our reduction can be stated as follows.

▶ **Proposition 8.** *Let B be a length-n width-3 ordered branching program (abbreviated **3OBP**), $m \geq k$, and $k \in [n]$ with the first and last layers having width at most 2. Then*

$$L^k(B) \leq n \cdot \binom{m}{k} \sum_{\ell \geq 0} 2^{-\ell(m-k)} L^k(D^{6(\ell+1)k})$$

where each $D^{6(\ell+1)k} = D_1^{6(\ell+1)k} \circ D_2^{6(\ell+1)k} \circ \cdots \circ D_r^{6(\ell+1)k}$, where $r \in [n]$, each $D_i^{6(\ell+1)k}$ is a 3OBP with at most $6(\ell+1)k$ non-regular layers, and the first and last layers of each D_i have width at most 2.

In Section 3.2, we will prove $L^k(D^{6(\ell+1)k}) \leq n \cdot O(\ell)^k$. Taking $m = 2k$, this implies $L^k(B) \leq n^2 \cdot O(k)^k$. Finally, we show that we may assume $k \leq O(\log n)$, so we get a Fourier growth bound of $L^k(B) \leq n^2 \cdot O(\log n)^k$. Here we focus on the proof of Proposition 8.

Define a **chunk** to be a 3OBP with exactly one non-regular layer. An ***l*-chunk** 3OBP is a 3OBP B such that $B = C_1 \circ C_2 \circ \cdots \circ C_l$, where each C_i is a chunk. Equivalently, an l-chunk 3OBP is a 3OBP with exactly l non-regular layers. The partitioning of B into chunks is not necessarily unique. But we fix one such partitioning for each 3OBP and simply

refer to the i^{th} chunk C_i. If B is an l-chunk length-n 3OBP, let $c_i \subset [n]$ be the coordinates corresponding to C_i.

We will compute a bound on the level-k Fourier weight of B via a series of "interwoven" restrictions similar to Steinberger's technique [36]. Lemma 9 below tells us that we may obtain a bound by bounding, in expectation, the level-k weight of a restricted branching program. We then argue that with high probability over this restriction, the width of the resulting program will be essentially reduced. In particular, there is a layer of width 2 after every $O(m)$ non-regular layers.

We now describe some notation that will be used for the interwoven restrictions. For $t \subset [m]$, define

$$g_t := \bigcup_{(i \bmod m)+1 \in t} c_i \quad \text{and} \quad G_t^k := \{s \subset g_t : |s| = k\}.$$

We refer to g_t as the **t^{th} group of indices** and G_t^k as the **t^{th} group of (order k) Fourier coefficients**. The following simple lemma tells us that we may bound the level-k Fourier weight by considering a fixed subset $t \subset [m]$ of size k and the level-k Fourier weight of the branching program that results by randomly restricting the variables outside of g_t:

▶ **Lemma 9.** *Let B be a length-n 3OBP, $k \in [n]$, $m \geq k$ and g_t as above. Then*

$$L^k(B) \leq \binom{m}{k} \max_{t \subset [m]:|t|=k} \mathop{\mathbb{E}}_{U} \left[L^k(B|_{\overline{g_t} \leftarrow U})\right].$$

Given Lemma 9, we may now prove Proposition 8 by giving an upper bound on $\mathop{\mathbb{E}}_{U} \left[L^k(B|_{\overline{g_t} \leftarrow U})\right]$ for a fixed $t \subset [m]$ with $|t| = k$. To do this, we prove that a random restriction to $\overline{g_t}$ will, with high probability, result in a branching program of the desired form.

▶ **Lemma 10.** *Let B be a length-n 3OBP, $k, \ell \in [n]$, $m \geq k$ and fix $t \subseteq [m]$ with $|t| = k$. Then with probability at least $1 - n \cdot 2^{-\ell(m-k)}$ over a random choice of $x \in \{0,1\}^n$,*

$$B|_{\overline{g_t} \leftarrow x} = D_1 \circ D_2 \circ \cdots \circ D_r,$$

where $r \in [n]$ and each D_i is a 3OBP with at most $6\ell k$ non-regular layers and the layer of vertices between D_{i-1} and D_i have width at most 2.

3.2 Part 2 – Mixing in Width-2

Now it remains to bound the Fourier mass of 3OBPs of the form given by Proposition 8.

▶ **Proposition 11.** *Let D^ℓ be a length-n 3OBP such that*

$$D^\ell = D_1^\ell \circ D_2^\ell \circ \cdots \circ D_r^\ell,$$

where each D_i^ℓ is a 3OBP with at most ℓ non-regular layers and width 2 in the first and last layers. Then $L^k(D^\ell) \leq 2n \cdot (6000(\ell+1))^k$ for all k.

A key notion in our proof is a measure of the extent to which a branching program (or subprogram) mixes, and the way this is reflected in the Fourier spectrum. For an ordered branching program D of width w, define

$$\lambda(D) = \max_{x \in \mathbb{R}^w : \sum_i x_i = 0} \frac{\left|\left| x \mathop{\mathbb{E}}_{U} [D[U]] \right|\right|_2}{||x||_2}.$$

The quantity $\lambda(D)$ is a measure of the **mixing** of D. If D is regular, we have $\lambda(D) \in [0,1]$, where 0 corresponds to perfect mixing and 1 to no mixing. If D is not regular, it is possible that $\lambda(D) > 1$. However, for width-2 – where $\mathop{\mathbb{E}}_{U}[D[U]]$ is a 2×2 matrix – it turns out that $\lambda(D) \leq 1$ even if D is non-regular. In particular,

$$\text{if } \mathop{\mathbb{E}}_{U}[D[U]] = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}, \quad \text{then } \lambda(D) = \frac{\left|\left|(1,-1)\mathop{\mathbb{E}}_{U}[D[U]]\right|\right|_2}{||(1,-1)||_2} = |1-\alpha-\beta|.$$

The rows of $\mathop{\mathbb{E}}_{U}[D[U]]$ must sum to 1 and have non-negative entries (as they are a probability distribution). So $\alpha, \beta \in [0,1]$, which implies $\lambda(D) \leq 1$. This fact is crucial to our analysis and is the main reason our results do not extend to higher widths.

Note that for any $s \neq 0$, the rows of $\widehat{D}[s]$ sum to zero. Thus for any branching program $D = D_1 \circ D_2$ and coefficient $\widehat{D}[s]$ with $s = (s_1, s_2)$ satisfying $s_2 = 0$, we have

$$\left|\left|\widehat{D}[s]\right|\right|_2 \leq \left|\left|\widehat{D_1}[s_1]\right|\right|_2 \cdot \lambda(D_2).$$

For branching programs B in which every layer is mixing – that is $\lambda(B_i) \leq C < 1$ for all i – this fact can be used with an inductive argument (simpler than the proof below) to obtain a $O(n)/(1-C)^{O(k)}$ bound on the level-k Fourier mass. We show that any D_i in the branching program of the form given by Proposition 8 will either mix well or have small Fourier mass after restriction. More precisely, define the **p-damped Fourier mass** of a branching program B as

$$L_p(B) = \sum_{k>0} p^k L^k(B) = \sum_{s \neq 0} p^{|s|} \left|\left|\widehat{B}[s]\right|\right|_2.$$

Note that $L^k(B) \leq L_p(B) p^{-k}$ for all k and p. The main lemma we prove in this section is the following.

▶ **Lemma 12.** *If D is a length-d 3OBP with $k \geq 1$ non-regular layers that has only two vertices in the first and last layers, then*

$$\lambda(D) + L_p(D) \leq 1$$

for any $p \leq 1/6000(k+1)$.

First, we show that Lemma 12 implies Proposition 11:

Proof of Proposition 11. We inductively show that

$$L_p(D_1^\ell \circ \cdots \circ D_i^\ell) \leq 2i,$$

and hence $L_p(D) \leq 2r \leq 2n$. For $i = 0$ this is trivial. Now suppose it holds for i. By decomposition (Lemma 6), we have

$$\begin{aligned} L_p(D_1^\ell \cdots D_i^\ell \circ D_{i+1}^\ell) \leq & L_p(D_1^\ell \cdots D_i^\ell) \cdot L_p(D_{i+1}^\ell) + L_p(D_1^\ell \cdots D_i^\ell)\lambda(D_{i+1}^\ell) \\ & + \left|\left|\widehat{D_1^\ell \cdots D_i^\ell}[0]\right|\right|_2 \cdot L_p(D_{i+1}^\ell) \\ \leq & L_p(D_1^\ell \cdots D_i^\ell) \cdot 1 + \sqrt{2} L_p(D_{i+1}^\ell) \leq 2i + 2. \end{aligned}$$

The first inequality follows from mixing and the second from Lemma 12. Thus, we have that $L^k(D^\ell) \leq p^{-k} L_p(D^\ell) \leq 2n \cdot (6000(\ell+1))^k$, as required ◀

Now we turn our attention to Lemma 12. We split into two cases: If $\lambda(D)$ is far from 1 i.e. $\lambda(D) \leq 0.99$, then we need only ensure $L_p(D) \leq 1/100$. This is the 'easy case' which proceeds much like the analysis of regular branching programs [32]. If $\lambda(D) = 1$, then D is trivial – i.e. $L_p(D) = 0$ – and we are also done. The 'hard case' is when $\lambda(D)$ is very close to 1. i.e. $0.99 \leq \lambda(D) < 1$.

Easy Case – Good Mixing

The argument used by Reingold et al. [32] for regular branching program can be extended to give the following.

▶ **Lemma 13.** *Let D be a 3OBP with at most k non-regular layers. If $p \leq 1/6000(k+1)$, then $L_p(D) \leq 1/100$.*

It immediately follows that $\lambda(D)+L_p(D) \leq 1$ when $p \leq 1/6000(k+1)$, assuming $\lambda(D) < 0.99$. This covers the 'easy' case of Lemma 12.

Hard Case – Poor Mixing

Now we consider the case where $\lambda(D) \in [0.99, 1]$.

▶ **Lemma 14.** *Let D be a 3OBP with at most k non-regular layers where the first and last layers of vertices have width 2. Suppose $\lambda(D) \in [0.99, 1]$. If $p \leq 1/(24k+12)$, then $L_p(D) + \lambda(D) \leq 1$.*

This covers the 'hard' case of Lemma 12 and, along with Lemma 13 completes the proof of Lemma 12.

Since D has width 2 in the first and last layers, we view $D[x]$ as a 2×2 matrix. We can write the expectation (which is stochastic) as

$$\mathop{\mathbb{E}}_{U}[D[U]] = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}.$$

We can assume (by permuting rows and columns) that $\lambda(D) = 1-\alpha-\beta$ and $\alpha, \beta \in [0, 1/100]$. Now write

$$D[x] = \begin{pmatrix} 1-f(x) & f(x) \\ g(x) & 1-g(x) \end{pmatrix},$$

where $f, g : \{0,1\}^d \to \{0,1\}$. Then $\alpha = \mathop{\mathbb{E}}_{U}[f(U)]$ and $\beta = \mathop{\mathbb{E}}_{U}[g(U)]$. We can view D has having two *corresponding* start and end states. The probability that, starting in the first start state, we end in the first end state is $1-\alpha \geq 0.99$. Likewise, the probability that, starting in the second start state, we end in the second end state is $1-\beta \geq 0.99$. The function f is computed by starting in the first start state and accepting if we end in the second end state – that is, if we "cross over". Likewise, g computes the function telling us whether we will cross over from the second start state to the first end state. Intuitively, there is a low (1/100) probability of crossing over, so the program behaves like two disjoint programs.

We will show that $L_p(f) \leq (12k+6)p\alpha$ and $L_p(g) \leq (12k+6)p\beta$ for $p \leq 1/(6k+3)$, from which the result follows by choosing p such that $L_p(f) \leq \alpha/2$ and $L_p(g) \leq \beta/2$.

The plan is as follows.

1. Show that we can partition the vertices of D into two sets with $O(k)$ edges crossing between the sets such that each layer has at least one vertex in each set. Intuitively, this partitions D into two width-2 branching programs with a few edges going between them.

2. Using this partition, show that we can write $f(x) = \sum_s \prod_j f_{s,j}(x_j)$, where each $f_{s,j}$ is a $\{0,1\}$-valued function computed by a *regular* width-2 branching program, the product is over $O(k)$ terms and the x_js are a partition of x.
3. Let $f_s(x) = \prod_j f_{s,j}(x_j)$ and $\alpha_s = \mathop{\mathbb{E}}_U[f_s(U)]$. Show that $L_p(f_s) \leq (12k+6)p\alpha_s$ for $p \leq 1/(6k+3)$. Then

$$L_p(f) \leq \sum_s L_p(f_s) \leq \sum_s (12k+6)p\alpha_s \leq (12k+6)p\alpha.$$

The same holds for g, which gives the result. Formally, these steps proceed as follows. See the full version of this paper for proofs of these lemmas.

Step 1

▶ **Lemma 15.** *Let D be a 3OBP with at most k non-regular layers and width-2 in the first and last layers of vertices. Suppose $\lambda(D) \in [0.99, 1]$. Then there is a partition of the vertices of D such that each layer has at least one vertex in each side of the partition and there are at most $2k+1$ layers with an edge that crosses the partition.*

There are two possible start states in the first layer. We partition the vertices based on whether they are more likely to be reached if we start at the first start state versus starting from the second start state. The $\lambda(D) \geq 0.99$ assumption tells us that this partition is very strong, in the sense that most vertices are much more likely to be visited from one start state than from the other. Consequently there are few edges crossing the partition.

Step 2

▶ **Lemma 16.** *Let D be a length-d 3OBP with at most k non-regular layers and width-2 in the first and last layers of vertices. Suppose $\lambda(D) \geq 0.99$. If $f : \{0,1\}^n \to \{0,1\}$ is the function computed by D, then we can write $f(x) = \sum_s \prod_j f_{s,j}(x_j)$, where each $f_{s,j}$ is computed by a regular width-2 ordered branching program and the x_j's are a partition of x into at most $6k+3$ parts.*

To prove this, we use the partition of Lemma 15. Intuitively, D is partitioned into two width-2 branching programs. The problem is the $O(k)$ layers where D is either non-regular or there is an edge crossing the partition – call these critical layers. We condition on what happens at the critical layers, which we can express with a width-2 program, and finally, we express f by summing over all possibilities for what happens in the critical layers. The product appears because we must use an AND that the event we are conditioning on is true.

Step 3

Now we have reduced the problem to analysing functions of a very simple form. We can use the basic properties of width-2 branching programs to prove the following, which suffices to prove Lemma 14.

▶ **Lemma 17.** *Let $f : \{0,1\}^n \to \{0,1\}$ be of the form $f(x) = \prod_{j \in [k]} f_j(x_j)$, where the x_js are a partition of x and each f_j is computed by a width-2 ordered regular branching program. Then $L_p(f) \leq 2kp \cdot \mathop{\mathbb{E}}_U[f(U)]$ for any $p \leq 1/k$.*

4 The Pseudorandom Generator

Our main result Theorem 1 follows from plugging our Fourier growth bound (Theorem 4) into the analysis of [32]. We include a general statement here for completeness:

▶ **Theorem 18.** *Let $\mathcal{C}$ be a set of ordered branching programs of length at most n and width at most w that is closed under restrictions and subprograms – that is, if $B \in \mathcal{C}$, then $B|_{t \leftarrow x} \in \mathcal{C}$ for all t and x and $B_{i \cdots j} \in \mathcal{C}$ for all i and j. Suppose that, for all $B \in \mathcal{C}$ and $k \in [n]$, we have $L^k(B) \leq ab^k$, where $b \geq 2$. Let $\varepsilon > 0$.*

Then there exists a pseudorandom generator $G_{a,b,n,\varepsilon} : \{0,1\}^{s_{a,b,n,\varepsilon}} \to \{0,1\}^n$ with seed length $s_{a,b,n,\varepsilon} = O\left(b \cdot \log(b) \cdot \log(n) \cdot \log\left(\frac{abwn}{\varepsilon}\right)\right)$ such that, for any length-n, width-w, read-once, oblivious (but unordered) branching program B that corresponds to an ordered branching program in $\mathcal{C}$,[5]

$$\left\| \mathop{\mathbb{E}}_{U_{s_{a,b,n,\varepsilon}}} \left[B[G_{a,b,n,\varepsilon}(U_{s_{a,b,n,\varepsilon}})]\right] - \mathop{\mathbb{E}}_{U}\left[B[U]\right] \right\|_2 \leq \varepsilon.$$

Moreover, $G_{a,b,n,\varepsilon}$ can be computed in space $O(s_{a,b,n,\varepsilon})$.

To prove Theorem 1 we set $\mathcal{C}$ to be the class of all 3OBPs of length at most n. Theorem 4 gives a bound corresponding to $a = \text{poly}(n)$ and $b = O(\log n)$. This gives the required generator. The statements of Theorems 1 and 18 differ in that Theorem 18 bounds the error of the pseudorandom generator with respect to a matrix-valued function, while Theorem 1 bounds the error with respect to a $\{0,1\}$-valued function. These statements are equivalent as the $\{0,1\}$-valued function is simply one entry in the matrix-valued function.

The pseudorandom generator is formally defined as follows.

Algorithm for $G_{a,b,n,\varepsilon} : \{0,1\}^{s_{a,b,n,\varepsilon}} \to \{0,1\}^n$.

Parameters: $n \in \mathbb{N}$, $\varepsilon > 0$.

Input: A random seed of length $s_{a,b,n,\varepsilon}$.

1. Compute appropriate values of $p \leq 1/2b$, $\varepsilon' = \varepsilon p/14w\log_2(n)$, $k \geq \log_2\left(8an^4w/\varepsilon'\right)$, $\delta \leq \varepsilon'(p/2)^{2k}$, and $\mu \leq \varepsilon'/2ab^k$. [6]
2. If $n \leq 320 \cdot \lceil \log_2(1/\varepsilon') \rceil / p$, output n truly random bits and stop.
3. Sample $T \in \{0,1\}^n$ where each bit has expectation p and the bits are δ-almost $2k$-wise independent.
4. If $|T| < pn/2$, output 0^n and stop.
5. Recursively sample $\tilde{U} \in \{0,1\}^{\lfloor n(1-p/2) \rfloor}$. i.e. $\tilde{U} = G_{a,b,\lfloor n(1-p/2) \rfloor,\varepsilon}(U)$.
6. Sample $X \in \{0,1\}^n$ from a μ-biased distribution.
7. Output $\text{Select}(T, X, \tilde{U}) \in \{0,1\}^n$.[7]

The analysis of this generator can be found in the full version of this paper.

---

[5] That is, there exists $B' \in \mathcal{C}$ and a permutation of the bits $\pi : \{0,1\}^n \to \{0,1\}^n$ such that $B[x] = B'[\pi(x)]$ for all x.

[6] For the purposes of the analysis we assume that ε', k, p, δ, and μ are the same at every level of recursion. So if $G_{a,b,n,w,\varepsilon}$ is being called recursively, use the same values of ε', p, k, δ, and μ as at the previous level of recursion. We pick values within a constant factor of these constraints.

[7] Technically, we must pad $\tilde{U}$ with zeros in the locations specified by T (i.e. $\tilde{U}_i = 0$ for $i \in T$) to obtain the right length.

5 Further Work

Our results hinge on the fact that "mixing" is well-understood for regular branching programs [8, 32, 26, 13, 37] and for (non-regular) width-2 branching programs [4]. We are able to use random restrictions to reduce from width 3 to width 2 (Section 3.1), where we can exploit our understanding of mixing (Section 3.2). Indeed, this understanding underpins most results for these restricted models of branching programs.

What about width 4 and beyond? Using a random restriction we can reduce analysing width 4 to "almost" width 3 – that is, Proposition 8 generalises. Unfortunately, the reduction does not give a true width-3 branching program and thus we cannot repeat the reduction to width 2. Moreover, we have a poor understanding of mixing for non-regular width-3 branching programs, which means we cannot use the same techniques that have worked for width-2 branching programs.

Our results provide some understanding of mixing in width-3. We hope this understanding can be developed further and will lead to proving Conjecture 3 and other results.

The biggest obstacle to extending our techniques to $w > 3$ is Lemma 12. The problem is that the parameter $\lambda(D)$ is no longer a useful measure of mixing for width-3 and above. In particular, $\lambda(D) > 1$ is possible if $\mathop{\mathbb{E}}_{U}[D[U]]$ is a 3×3 matrix. To extend our techniques, we need a better notion of mixing. Using $\lambda(D)$ is useful for regular branching programs (it equals the second eigenvalue for regular programs), but is of limited use for non-regular branching programs. Our proof uses a different notion of mixing – collisions: To prove Proposition 8, we used the fact that a random restriction of a non-regular layer will with probability at least $1/2$ result in the width of the right side of the layer being reduced. This is a form of mixing, but it is not captured by λ. Ideally, we want a notion of mixing that captures both λ and width-reduction under restrictions.

Our proofs combine the techniques of Braverman et al. [8] and those of Brody and Verbin [9] and Steinberger [36]. We would like to combine them more cleanly – presently the proof is split into two parts (Proposition 8 and Lemma 12). This would likely involve developing a deeper understanding of the notion of mixing.

Our seed length $\tilde{O}(\log^3 n)$ is far from the optimal $O(\log n)$. Further improvement would require some new techniques:

We could potentially relax our notion of Fourier growth to achieve better results. Rather than bounding $L^k(f)$, it suffices to bound $L^k(g)$, where g *approximates* f:

▶ **Proposition 19** ([14, Proposition 2.6]). *Let $f, f_+, f_- : \{0,1\}^n \to \mathbb{R}$ satisfy $f_-(x) \le f(x) \le f_+(x)$ for all x and $\mathop{\mathbb{E}}_{U}[f_+(U) - f_-(U)] \le \delta$. Then any ε-biased distribution X gives*

$$\left|\mathop{\mathbb{E}}_{X}[f(X)] - \mathop{\mathbb{E}}_{U}[f(U)]\right| \le \delta + \varepsilon \cdot \max\{L(f_+), L(f_-)\}.$$

The functions f_+ and f_- are called sandwiching polynomials for f. This notion of sandwiching is in fact a tight characterisation of small bias [14, Proposition 2.7]. That is, any function f fooled by all small bias generators has sandwiching polynomials satisfying the hypotheses of Proposition 19.

Gopalan et al. [17] use sandwiching polynomials in the analysis of their generator for CNFs. This allows them to set a constant fraction of the bits at each level of recursion ($p = \Omega(1)$), while we set a $1/O(\log n)$ fraction at each level. We would like to similarly exploit sandwiching polynomials for branching programs to improve the seed length of the generator.

A further avenue for improvement would be to modify the generator construction to have $\Theta(1/p)$ levels of recursion, rather than $\Theta(\log(n)/p)$. This would require a significantly different analysis.

References

1 Anil Ada, Omar Fawzi, and Hamed Hatami. Spectral norm of symmetric functions. In *APPROX-RANDOM*, pages 338–349. Springer, 2012.

2 Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. In *FOCS*, pages 544–553, 1990.

3 Michael Ben-Or and Nathan Linial. Collective coin flipping. *Randomness and Computation*, 5:91–115, 1990.

4 Andrej Bogdanov, Zeev Dvir, Elad Verbin, and Amir Yehudayoff. Pseudorandomness for width 2 branching programs. *ECCC*, 16:70, 2009.

5 Andrej Bogdanov, Periklis A. Papakonstantinou, and Andrew Wan. Pseudorandomness for read-once formulas. In *FOCS*, pages 240–246, 2011.

6 Jean Bourgain. On the distribution of the fourier spectrum of boolean functions. *Israel J. Mathematics*, 131(1):269–276, 2002.

7 Yigal Brandman, Alon Orlitsky, and John L. Hennessy. A spectral lower bound techniqye for the size of decision trees and two level and/or circuits. *IEEE Transactions on Computers*, 39(2):282–287, 1990.

8 Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *FOCS*, pages 40–47, 2010.

9 Joshua Brody and Elad Verbin. The coin problem and pseudorandomness for branching programs. In *FOCS*, pages 30–39, 2010.

10 Jehoshua Bruck. Harmonic analysis of polynomial threshold functions. *SIAM J. Discrete Mathematics*, 3:168–177, 1990.

11 Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions, and spectral norms. *SIAM J. Computing*, 21(1):33–42, 1992.

12 L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. In *FOCS*, pages 599–608, 2011.

13 Anindya De. Pseudorandomness for permutation and regular branching programs. In *CCC*, pages 221–231, 2011.

14 Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 6302 of *Lecture Notes in Computer Science*, pages 504–517. Springer, 2010.

15 Lars Engebretsen, Piotr Indyk, and Ryan O'Donnell. Derandomized dimensionality reduction with applications. In *SODA*, pages 705–712, 2002.

16 Ehud Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

17 Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil Vadhan. Better pseudorandom generators from milder pseudorandom restrictions. In *FOCS*, pages 120–129, 2012.

18 Ben Green and Tom Sanders. Boolean functions with small spectral norm. *Geometric and Functional Analysis*, 18(1):144–162, 2008.

19 Vince Grolmusz. On the power of circuits with gates of low ℓ_1 norms. *Theoretical computer science*, 188(1):117–128, 1997.

20 Iftach Haitner, Danny Harnik, and Omer Reingold. On the power of the randomized iterate. In *CRYPTO*, 2006.

21 Alexander Healy, Salil Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. *SIAM J. Computing*, 35(4):903–931 (electronic), 2006.

22 R. Impagliazzo, R. Meka, and D. Zuckerman. Pseudorandomness from shrinkage. In *FOCS*, pages 111–119, 2012.

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

24 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

25 Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. In *APPROX-RANDOM*, pages 354–365, 2005.

26 Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products. In *STOC*, pages 263–272, 2011.

27 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Computing*, 22(6):1331–1348, 1993.

28 Yishay Mansour. An $O(n \log \log n)$ learning algorithm for DNF under the uniform distribution. *J. CSS*, 50(3):543–550, 1995.

29 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Computing*, 22:838–856, 1993.

30 Noam Nisan. $\mathcal{RL} \subset \mathcal{SC}$. In *STOC*, pages 619–623, 1992.

31 Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In *STOC*, pages 235–244, 1993.

32 Omer Reingold, Thomas Steinke, and Salil Vadhan. Pseudorandomness for regular branching programs via fourier analysis. In *APPROX-RANDOM*, pages 655–670, 2013.

33 Michael Saks and Shiyu Zhou. $\mathrm{BP_HSPACE}(S) \subset \mathrm{DSPACE}(S^{3/2})$. *J. CSS*, 58(2):376–403, 1999.

34 Amir Shpilka, Avishay Tal, et al. On the structure of boolean functions with small spectral norm. In *ITCS*, pages 37–48, 2014.

35 D. Sivakumar. Algorithmic derandomization via complexity theory. In *CCC*, page 10, 2002.

36 John Steinberger. The distinguishability of product distributions by read-once branching programs. In *CCC*, pages 248–254, 2013.

37 Thomas Steinke. Pseudorandomness for permutation branching programs without the group theory. *ECCC*, 19:83, 2012.

38 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *FOCS*, pages 658–667, 2013.

39 Jiří Šíma and Stanislav Žák. A sufficient condition for sets hitting the class of read-once branching programs of width 3. In *SOFSEM*, pages 406–418, 2012.