Pseudorandomness for Regular Branching Programs via Fourier Analysis

Omer Reingold[1], Thomas Steinke[2,⋆], and Salil Vadhan[2,⋆⋆]

[1] Microsoft Research Silicon Valley, Mountain View, CA
Omer.Reingold@microsoft.com

[2] School of Engineering and Applied Sciences, Harvard University, Cambridge MA
{tsteinke,salil}@seas.harvard.edu

Abstract. We present an explicit pseudorandom generator for oblivious, read-once, permutation branching programs of constant width that can read their input bits in any order. The seed length is $O(\log^2 n)$, where n is the length of the branching program. The previous best seed length known for this model was $n^{1/2+o(1)}$, which follows as a special case of a generator due to Impagliazzo, Meka, and Zuckerman (FOCS 2012) (which gives a seed length of $s^{1/2+o(1)}$ for arbitrary branching programs of size s). Our techniques also give seed length $n^{1/2+o(1)}$ for general oblivious, read-once branching programs of width $2^{n^{o(1)}}$, which is incomparable to the results of Impagliazzo et al.

Our pseudorandom generator is similar to the one used by Gopalan et al. (FOCS 2012) for read-once CNFs, but the analysis is quite different; ours is based on Fourier analysis of branching programs. In particular, we show that an oblivious, read-once, *regular* branching program of width w has Fourier mass at most $(2w^2)^k$ at level k, independent of the length of the program.

1 Introduction

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 pseudorandom generator 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 [1], deterministic dimension reduction and SDP rounding [2], hashing [3], hardness

⋆ Work done in part while at Stanford University. Supported by NSF grant CCF-1116616 and the Lord Rutherford Memorial Research Fellowship.

⋆⋆ Work done in part when on leave as a Visiting Researcher at Microsoft Research Silicon Valley and a Visiting Scholar at Stanford University. Supported in part by NSF grant CCF-1116616 and US-Israel BSF grant 2010196.

amplification [4], almost k-wise independent permutations [5], and cryptographic pseudorandom generator constructions [6].

Unfortunately, for fooling general logspace algorithms, there has been essentially no improvement since the classic work of Nisan [7], which provided a pseudorandom generator of seed length $O(\log^2 n)$. Instead, a variety of works have improved the seed length for various restricted classes of logspace algorithms, such as algorithms that use $n^{o(1)}$ random bits [8, 9], combinatorial rectangles [10–13] random walks on graphs [14, 15], branching programs of width 2 or 3 [16–18], and regular or permutation branching programs (of bounded width) [19–23].

The vast majority of these works are based on Nisan's generator or its variants by Impagliazzo, Nisan, and Wigderson [24] and Nisan and Zuckerman [8], and show how the analysis (and hence the final parameters) of these generators can be improved for logspace algorithms that satisfy the additional restrictions. All three of these generators are based on recursive use of the following principle: if we consider two consecutive time intervals I_1, I_2 in a space s computation and use some randomness r to generate the pseudorandom bits fed to the algorithm during interval I_1, then at the start of I_2, the algorithm will 'remember' at most s bits of information about r. So we can use a randomness extractor to extract roughly $|r|-s$ almost uniform bits from r (while investing only a small additional amount of randomness for the extraction). This paradigm seems unlikely to yield pseudorandom generators for general logspace computations that have a seed length of $\log^{1.99} n$ (see [20]).

Thus, there is a real need for a different approach to constructing pseudorandom generators for space-bounded computation. One new approach has been suggested in the recent work of Gopalan et al. [25], which constructed improved pseudorandom generators for read-once CNF formulas and combinatorial rectangles, and hitting set generators for width 3 branching programs. Their basic generator (e.g. for read-once CNF formulas) works as follows: Instead of considering a fixed partition of the bits into intervals, they pseudorandomly partition the bits into two groups, assign the bits in one group using a small-bias generator [26], and then recursively generate bits for the second group. While it would not work to assign *all* the bits using a single sample from a small-bias generator, it turns out that generating a pseudorandom partial assignment is a significantly easier task.

An added feature of the Gopalan et al. generator is that its pseudorandomness properties are independent of the order in which the output bits are read by a potential distinguisher. In contrast, Nisan's generator and its variants depend heavily on the ordering of bits (the intervals I_1 and I_2 above cannot be interleaved), and in fact it is known that a particular instantiation of Nisan's generator fails to be pseudorandom if the (space-bounded) distinguisher can read the bits in a different order [27, Corollary 3.18]. Recent works [28, 29] have constructed nontrivial pseudorandom generators for space-bounded algorithms that can read their bits in any order, but the seed length achieved is larger than $\sqrt{n}$.

In light of the above, a natural question is whether the approach of Gopalan et al. can be extended to a wider class of space-bounded algorithms. We make progress on this question by using the same approach to construct a pseudorandom generator with seed length $O(\log^2 n)$ for constant-width, read-once, oblivious permutation branching programs that can read their bits in any order. In analysing our generator, we develop new Fourier-analytic tools for proving pseudorandomness against space-bounded algorithms.

1.1 Models of Space-Bounded Computation

A (layered) **branching program** B is a nonuniform model of space-bounded computation. The program maintains a **state** from the set $[w] = \{1, \ldots, w\}$ and, at each time step i, reads one bit of its input $x \in \{0,1\}^n$ and updates its state according to a transition function $B_i : \{0,1\} \times [w] \to [w]$. The parameter w is called the **width** of the program, and corresponds to a space bound of $\log w$ bits. We allow the transition function B_i to be different at each time step i. We consider several restricted forms of branching programs:

- **Read-once branching programs** read each input bit at most once.
- **Oblivious branching programs** choose which input bit to read depending only on the time step i, and not on the current state
- **Ordered branching programs** (a.k.a. streaming algorithms) always read input bit i in time step i (hence are necessarily both read-once and oblivious).

To derandomize randomized space-bounded computations (e.g. prove RL = L), it suffices to construct pseudorandom generators that fool ordered branching programs of polynomial width ($w = \text{poly}(n)$), and hence this is the model addressed by most previous constructions (including Nisan's generator). However, the more general models of oblivious and read-once branching programs are also natural to study, and, as discussed above, can spark the development of new techniques for reasoning about pseudorandomness.

As mentioned earlier, Nisan's pseudorandom generator [7] achieves $O(\log^2 n)$ seed length for *ordered* branching programs of polynomial width. It is known how to achieve $O(\log n)$ seed length for ordered branching programs width 2 [17], and for width 3, it is only known how to construct "hitting-set generators" (a weaker form of pseudorandom generators) with seed length $O(\log n)$ [18, 25]. (The seed length is $\tilde{O}(\log n)$ if we want the error of the hitting set generator to be subconstant.) For pseudorandom generators for width $w \geq 3$ and hitting-set generators for width $w \geq 4$, there is no known construction with seed length $o(\log^2 n)$.

The study of pseudorandomness against non-ordered branching programs started more recently. Tzur [27] showed that there are oblivious, read-once, constant-width branching programs that can distinguish the output of Nisan's generator from uniform. Bogdanov, Papakonstantinou, and Wan [28] exhibited a pseudorandom generator with seed length $(1 - \Omega(1)) \cdot n$ for oblivious read-once branching programs of width w for $w = 2^{\Omega(n)}$. Impagliazzo, Meka, and Zuckerman [29] gave a pseudorandom generator with seed length $s^{1/2+o(1)}$ for arbitrary branching programs of size s; note that $s = O(nw)$ for a read-once branching program of width w and length n.

We consider two further restrictions on branching programs:

- **Regular branching programs** are oblivious branching programs with the property that, if the distribution on states in any layer is uniformly random and the input bit read by the program at that layer is uniformly random, then the resulting distribution on states in the next layer is uniformly random. This is equivalent to requiring that the bipartite graph associated with each layer of the program, where we have edges from each state $u \in [w]$ in layer i to the possible next-states $u_0, u_1 \in [w]$ in layer $i+1$ (if the input bit is b, the state goes to u_b), is a regular graph.
- **Permutation branching programs** are a further restriction, where we require that for each setting of the input string, the mappings between layers are permutations. This is equivalent to saying that (regular) bipartite graphs corresponding to each layer are decomposed into two perfect matchings, one corresponding to each value of the current input bit being read.

The fact that pseudorandomness for permutation branching programs might be easier than for general branching programs was suggested by the proof that Undirected S-T Connectivity is in Logspace [14] and its follow-ups [15, 30]. Specifically, the latter works construct "pseudorandom walk generators" for "consistently labelled" graphs. Interpreted for permutation branching programs, these results ensure that if an ordered permutation branching program has the property that every layer has a nonnegligible amount of "mixing" — meaning that the distribution on states becomes closer to uniform, on a truly random input — then the overall program will also have mixing when run on the output of the pseudorandom generator (albeit at a slower rate). The generator has a seed length of $O(\log n)$ even for ordered permutation branching programs of width $\mathrm{poly}(n)$. Reingold, Trevisan, and Vadhan [15] also show that if a generator with similar properties could be constructed for (ordered) regular branching programs of polynomial width, then this would suffice to prove RL = L. Thus, in the case of polynomial width, regularity is not a significant constraint.

Recently, there has been substantial progress on constructing pseudorandom generators for ordered regular and permutation branching programs of constant width. Braverman, Rao, Raz, and Yehudayoff [19] and Brody and Verbin [20] gave pseudorandom generators with seed length $\tilde{O}(\log n)$ for ordered regular branching programs of constant width. Koucký, Nimbhorkar and Pudlák [21] showed that the seed length could be further improved to $O(\log n)$ for ordered, permutation branching programs of constant width; see [22, 23] for simplifications and improvements.

All of these generators for ordered regular and permutation branching programs are based on refined analyses of the pseudorandom generator construction of Impagliazzo, Nisan, and Wigderson [24].

1.2 Our Results and Techniques

Our main result is a pseudorandom generator for read-once, oblivious, (unordered) permutation branching programs of constant width:

Theorem 1.1 (Main Result). *For every constant w, there is an explicit pseudorandom generator $G : \{0,1\}^{O(\log^2 n)} \to \{0,1\}^n$ fooling oblivious, read-once (but unordered), permutation branching programs of width w and length n.*

To be precise, the seed length and space complexity of the pseudorandom generator is

$$O(w^2 \log(w) \log(n) \log(nw/\varepsilon) + w^4 \log^2(w/\varepsilon))$$

for oblivious, read-once, permutation branching programs of length n and width w, where ε is the error.

Previously, it was only known how to achieve a seed length of $n^{1/2+o(1)}$ for this model, as follows from the aforementioned results of Impagliazzo, Meka, and Zuckerman [29] (which actually holds for arbitrary branching programs).

Our techniques also achieve seed length $n^{1/2+o(1)}$ for arbitrary read-once, oblivious branching programs of width up to $2^{n^{o(1)}}$:

Theorem 1.2. *There is an explicit pseudorandom generator $G : \{0,1\}^{\tilde{O}(\sqrt{n} \log w)} \to \{0,1\}^n$ fooling oblivious, read-once (but unordered) branching programs of width w and length n.*

This result is incomparable to that of Impagliazzo et al. [29]. Their seed length depends polynomially on the width w, so require width $w = n^{o(1)}$ to achieve seed length $n^{1/2+o(1)}$. On the other hand, our result is restricted to *read-once, oblivious* branching programs.

Our construction of the generator in Theorem 1.1 is essentially the same as the generator of Gopalan et al. [25] for read-once CNF formulas, but with a new analysis (and different setting of parameters) for read-once, oblivious, permutation branching programs. The generator works by selecting a subset $T \subset [n]$ of output coordinates in a pseudorandom way, assigning the bits in T using another pseudorandom distribution X, and then recursively assigning the bits outside T. We generate T using an almost $O(\log n)$-wise independent distribution, including each coordinate $i \in T$ with a constant probability p_w depending only on the width w. We assign the bits in T using a small-bias distribution X on $\{0,1\}^n$ [26]; such a generator has the property that for every nonempty subset $S \subset [n]$, the parity $\oplus_{i \in S} X_i$ of bits in S has bias at most ε. Generating T requires $O(\log n)$ random bits, generating X requires $O(\log n)$ bits (even for $\varepsilon = 1/\text{poly}(n)$), and we need $O(\log n)$ levels of recursion to assign all the bits. This gives us our $O(\log^2 n)$ seed length.

Let $B : \{0,1\}^n \to \{0,1\}$ be a function computed by an oblivious, read-once, permutation branching program of width w. Following [25], to show that our pseudorandom generator fools B, it suffices to show that the partial assignment generated in a single level of recursion approximately preserves the acceptance probability of B (on average). To make this precise, we need a bit of notation.

For a set $t \subset [n]$, a string $x \in \{0,1\}^n$, and $y \in \{0,1\}^{n-|t|}$, define $\mathrm{Select}(t,x,y) \in \{0,1\}^n$ as follows:

$$\mathrm{Select}(t,x,y)_i = \begin{cases} x_i & \text{if } i \in t \\ y_{|\{j \le i : j \notin t\}|} & \text{if } i \notin t \end{cases}$$

Once we choose a set $t \leftarrow T$ and an assignment $x \leftarrow X$ to the variables in t, the residual acceptance probability of B is $\mathbb{P}_{U}[B(\mathrm{Select}(t,x,U)) = 1]$, where U is the uniform distribution on $\{0,1\}^n$. So, the average acceptance probability over $t \leftarrow T$ and $x \leftarrow X$ is $\mathbb{P}_{T,X,U}[B(\mathrm{Select}(T,X,U)) = 1]$. We would like this to be close to the acceptance probability under uniformly random bits, namely $\mathbb{P}_{U}[B(U) = 1] = \mathbb{P}_{T,U',U}[B(\mathrm{Select}(T,U',U) = 1]$. That is, we would like our small-bias distribution X to fool the function $B'(x) := \mathbb{E}_{T,U}[B(\mathrm{Select}(T,x,U))]$. The key insight in [25] is that B' can be a significantly easier function to fool than B, and even than fixed restrictions of B (like $B(\mathrm{Select}(t,\cdot,y))$ for fixed t and y). We show that the same phenomenon holds for oblivious, read-once, *regular* branching programs. (The reason that the analysis of our overall pseudorandom generator applies only for *permutation* branching programs is that regularity is not preserved under restriction (as needed for the recursion), whereas the permutation property is.)

To show that a small-bias space fools $B'(x)$, it suffices to show that the **Fourier mass** of B', namely $\sum_{s \in \{0,1\}^n, s \neq 0} |\widehat{B'}[s]|$, is bounded by $\mathrm{poly}(n)$. (Here $\widehat{B'}[s] = \mathbb{E}_{U}\left[B'[U] \cdot (-1)^{s \cdot U}\right]$ is the standard Fourier transform over $\mathbb{Z}_2^n$. So $\widehat{B'}[s]$ measures the correlation of B' with the parity function defined by s.) We show that this is indeed the case (for most choices of the set $t \leftarrow T$):

Theorem 1.3 (Main Lemma). *For every constant w, there are constants $p_w > 0$ and $d_w \in \mathbb{N}$ such that the following holds. Let $B : \{0,1\}^n \to \{0,1\}$ be computed by an oblivious, read-once, regular branching program of width w and length $n \geq d_w$. Let $T \subset [n]$ be a randomly chosen set so that every coordinate $i \in [n]$ is placed in T with probability p_w and these choices are n^{-d_w}-almost $(d_w \log n)$-wise independent. Then with high probability over $t \leftarrow T$ $B'(x) = \mathbb{E}_{U}[B(\mathrm{Select}(t,x,U))]$ has Fourier mass at most n^{d_w}.*

As a warm-up, we begin by analysing the Fourier mass in the case the set T is chosen completely at random, with every coordinate included independently with probability p_w. In this case, it is more convenient to average over T and work with $B'(x) = \mathbb{E}_{T,U}[B(\mathrm{Select}(T,x,U))]$. Then it turns out that $\widehat{B'}[s] = p_w^{|s|} \cdot \widehat{B}[s]$, where $|s|$ denotes the Hamming weight of the vector s. Thus, it suffices to analyse the original program B and show that for each $k \in \{1, \cdots, n\}$, the Fourier mass of B restricted to s of weight k is at most c_w^k, where c_w is a constant depending only on w (not on n). We prove that this is indeed the case for regular branching programs:

Theorem 1.4. *Let $B : \{0,1\}^n \to \{0,1\}$ be a function computed by an oblivious, read-once, regular branching program of width w. Then for every $k \in \{1, \ldots, n\}$, we have*

$$\sum_{s \in \{0,1\}^n : |s| = k} |\widehat{B}[s]| \leq (2w^2)^k.$$

Our proof of Theorem 1.4 relies on the main lemma of Braverman et al. [19], which intuitively says that in a bounded-width, read-once, oblivious, regular branching program, only a constant number of bits have a significant effect on the acceptance probability. More formally, if we sum, for every time step i and all possible states v at time i, the absolute difference between the acceptance probability after reading a 0 versus reading a 1 from state v, the total will be bounded by $\text{poly}(w)$ (independent of n). This directly implies a bound of $\text{poly}(w)$ on the Fourier mass of B at the first level: the correlation of B with a parity of weight 1 is bounded by the effect of a single bit on the output of B. We then bound the correlation of B with a parity of weight k by the correlation of a *prefix* of B with a parity of weight $k-1$ times the effect of the remaining bit on B. Thus we inductively obtain the bound on the Fourier mass of B at level k.

Our proof of Theorem 1.3 for the case of a pseudorandom restriction T uses the fact that we can decompose the high-order Fourier coefficients of an oblivious, read-once branching program B' into products of low-order Fourier coefficients of "subprograms" (intervals of consecutive layers) of B'. Using an almost $O(\log n)$-wise independent choice of T enables us to control the Fourier mass at level $O(\log n)$ for all subprograms of B', which suffices to control the total Fourier mass of B'.

2 Preliminaries

2.1 Branching Programs

We define a length-n, width-w **program** to be 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 $B[x](u)$.

In our applications, the input x is randomly (or pseudorandomly) chosen, in which case a program can be viewed as a Markov chain randomly taking initial states to final states. For each $x \in \{0,1\}^n$, we let $B[x] \in \{0,1\}^{w \times w}$ be a matrix defined by

$$B[x](u,v) = 1 \iff B[x](u) = v.$$

For a random variable X on $\{0,1\}^n$, we have $\mathop{\mathbb{E}}_X[B[X]] \in [0,1]^{w \times w}$, where $\mathop{\mathbb{E}}_R[f(R)]$ is the **expectation** of a function f with respect to a random variable R. Then the entry in the uth row and vth 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.

A branching program reads one bit of the input at a time (rather than reading x all at once) maintaining only a state in $[w] = \{1, 2, \cdots, w\}$ at each step.

We capture this restriction by demanding that the program be composed of several smaller programs, as follows.

Let B and B' be width-w programs of length n and n' respectively. We define the **concatenation** $B \circ B' : \{0,1\}^{n+n'} \times [w] \to [w]$ of B and B' by

$$(B \circ B')[x \circ x'](u) := B'[x'](B[x](u)),$$

which is a width-w, length-$(n+n')$ program. That is, we run B and B' on separate inputs, but the final state of B becomes the start state of B'. Concatenation corresponds to matrix multiplication—that is, $(B \circ B')[x \circ x'] = B[x] \cdot B'[x']$, where the two programs are concatenated on the left hand side and the two matrices are multiplied on the right hand side.

A length-n, width-w, **ordered branching program** 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 $B_{i\cdots j} := B_i \circ B_{i+1} \circ \cdots \circ B_j$.

General read-once, oblivious branching programs (a.k.a. unordered branching programs) can be reduced to the ordered case by a permutation of the input bits. Formally, a **read-once, oblivious branching program** B is an ordered branching program B' composed with a permutation π. That is, $B[x] = B'[\pi(x)]$, where the i^{th} bit of $\pi(x)$ is the $\pi(i)^{\text{th}}$ bit of x.

For a program B and an arbitrary distribution X, the matrix $\underset{X}{\mathbb{E}}[B[X]]$ is **stochastic**—that is, $\sum_v \underset{X}{\mathbb{E}}[B[X]](u,v) = 1$ for all u and $\underset{X}{\mathbb{E}}[B[X]](u,v) \geq 0$ for all u and v. A program B is called a **regular program** if the matrix $\underset{U}{\mathbb{E}}[B[U]]$ is **doubly stochastic**—that is, both $\underset{U}{\mathbb{E}}[B[U]]$ and its transpose $\underset{U}{\mathbb{E}}[B[U]]^*$ are stochastic. A program B is called a **permutation program** if $B[x]$ is a permutation matrix for every x or, equivalently, $B[x]$ is doubly stochastic. Note that a permutation program is necessarily a regular program and, if both B and B' are regular or permutation programs, then so is their concatenation.

A regular program B has the property that the uniform distribution is a stationary distribution of the Markov chain $\underset{U}{\mathbb{E}}[B[U]]$, whereas, if B is a permutation program, the uniform distribution is stationary for $\underset{X}{\mathbb{E}}[B[X]]$ for *any* X.

A **regular branching program** is a branching program where each layer B_i is a regular program and likewise for a **permutation branching program**.

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

$$\widehat{B}[s] := \underset{U}{\mathbb{E}}[B[U]\chi_s(U)],$$

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. The **order** of a Fourier coefficient $\widehat{B}[s]$ is $|s|$—the **Hamming weight** of s, which is the size of the set s or the number of 1s in the string s. Note that this is equivalent to taking the real-valued Fourier transform of each of the w^2 entries of B separately, but we will see below that this matrix-valued Fourier transform is nicely compatible with matrix algebra.

For a random variable X over $\{0,1\}^n$ we define its s^{th} **Fourier coefficient** as

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

which, up to scaling, is the same as taking the real-valued Fourier transform of the probability mass function of X. We have the following useful properties.

Lemma 2.1. *Let $A, B : \{0,1\}^n \to \mathbb{R}^{w \times w}$ 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 $\widehat{C}[s \circ t] = \widehat{A}[s] \cdot \widehat{B}[t]$.*
- *Expectation: $\mathop{\mathbb{E}}_{X}\left[B[X]\right] = \sum_s \widehat{B}[s]\widehat{X}(s)$.*
- *Parseval's Identity: $\sum_{s \in \{0,1\}^n} \left|\left|\widehat{B}[s]\right|\right|_{Fr}^2 = \mathop{\mathbb{E}}_{U}\left[||B[U]||_{Fr}^2\right]$, where $||\cdot||_{Fr}$ is the Frobenius norm.*

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 in [28]. If B is a length-n width-w branching program, then, for all $s \in \{0,1\}^n$,

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

2.3 Fourier Mass

Define the **Fourier mass** of a matrix-valued function B to be

$$L_2(B) := \sum_{s \neq 0} \left|\left|\widehat{B}[s]\right|\right|_2,$$

where $||M||_2 := \max_x ||xM||_2 / ||x||_2$ is the **spectral norm**. Also, define the **Fourier mass of B at level k** as

$$L_2^k(B) := \sum_{s \in \{0,1\}^n : |s| = k} \left|\left|\widehat{B}[s]\right|\right|_2.$$

Note that $L_2(B) = \sum_{k \geq 1} L_2^k(B)$.

The Fourier mass is unaffected by order:

Lemma 2.2. *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, $L_2(B) = L_2(B')$ and $L_2^k(B) = L_2^k(B')$ for all k.*

Lemma 2.2 implies that the Fourier mass of any read-once, oblivious branching program is equal to the Fourier mass of the corresponding ordered branching program.

A random variable X is called ε-**biased** if $|\hat{X}[s]| \le \varepsilon$ for all $s \ne 0^n$ [26]. If $L_2(B)$ is small, then B is fooled by any small-bias distribution:

Lemma 2.3. *Let B be a length-n, width-w, branching program. Let X be a ε-biased random variable on $\{0,1\}^n$. We have*

$$\left|\left| \mathop{\mathbb{E}}_{X}[B[X]] - \mathop{\mathbb{E}}_{U}[B[U]] \right|\right|_2 = \left|\left| \sum_{s \ne 0} \widehat{B}[s]\widehat{X}(s) \right|\right|_2 \le L_2(B)\varepsilon.$$

In the worst case $L_2(B) = 2^{\Theta(n)}$, even for a length-n width-3 permutation branching program B. For example, the program $B_{\text{mod } 3}$ that computes the Hamming weight of its input modulo 3 has exponential Fourier mass.

We show that, using 'restrictions', we can ensure that $L_2(B)$ is small.

3 Fourier Analysis of Regular Branching Programs

We use a result by Braverman et al. [19]. The following is a Fourier-analytic reformulation of their result.

Lemma 3.1 ([19, Lemma 4]). *Let B be a length-n, width-w, ordered, regular branching program. Then*

$$\sum_{1 \le i \le n} \left|\left| \widehat{B_{i \cdots n}}[1 \circ 0^{n-i}] \right|\right|_2 \le 2w^2.$$

Braverman et al. instead consider the sum, over all $i \in [n]$ and all states $u \in [w]$ at layer i, of the difference in acceptance probabilities if we run the program starting at v with a 0 followed by random bits versus a 1 followed by random bits. They refer to this quantity as the **weight** of B. Their result can be expressed in Fourier-analytic terms by considering subprograms $B_{i \cdots n}$ that are the original program with the first $i-1$ layers removed:

$$\sum_{1 \le i \le n} \left|\left| \widehat{B_{i \cdots n}}[1 \circ 0^{n-i}]q \right|\right|_1 \le 2(w-1)$$

for any $q \in \{0,1\}^w$ with $\sum_u q(u) = 1$. (The vector q can be used to specify the accept state of B, and the v^{th} row of $\widehat{B_{i \cdots n}}[1 \circ 0^{n-i}]q$ is precisely the difference in acceptance probabilities mentioned above.) By summing over all w possible q, we obtain

$$\sum_{i\in[n]}\sum_{u}\left|\left|\widehat{B_{i\cdots n}}[1\circ 0^{n-i}](\cdot,u)\right|\right|_1 \leq 2w(w-1).$$

This implies Lemma 3.1, as the spectral norm of a matrix is bounded by the sum of the 1-norms of the columns.

Lemma 3.1 is similar (but not identical) to a bound on the first-order Fourier coefficients of a regular branching program: The term $\widehat{B_{i\cdots n}}[1\circ 0^{n-i}]$ measures the effect of the i^{th} bit on the output of B when we start the program at layer i, whereas the i^{th} first-order Fourier coefficient $\widehat{B}[0^{i-1}\circ 1\circ 0^{n-i}]$ measures the effect of the i^{th} bit when we start at the first layer and run the first $i-1$ layers with random bits. This difference allows us to use Lemma 3.1 to obtain a bound on all low-order Fourier coefficients of a regular branching program:

Theorem 3.2. *Let B be a length-n, width-w, read-once, oblivious, regular branching program. Then*

$$L_2^k(B) := \sum_{s\in\{0,1\}^n:|s|=k}\left|\left|\widehat{B}[s]\right|\right|_2 \leq (2w^2)^k.$$

The bound does not depend on n, even though we are summing $\binom{n}{k}$ terms.

Proof. By Lemma 2.2, we may assume that B is ordered. We perform an induction on k. If $k=0$, then there is only one Fourier coefficient to bound—namely, $\widehat{B}[0^n] = \underset{U}{\mathbb{E}}[B[U]]$, which is doubly stochastic. The base case follows from the fact that every doubly stochastic matrix has spectral norm 1. Suppose the result holds for k. We split the Fourier coefficients based on where the last 1 is:

$$\begin{aligned}
&\sum_{s\in\{0,1\}^n:|s|=k+1}\left|\left|\widehat{B}[s]\right|\right|_2\\
&\quad=\sum_{1\leq i\leq n}\;\sum_{s\in\{0,1\}^{i-1}:|s|=k}\left|\left|\widehat{B}[s\circ 1\circ 0^{n-i}]\right|\right|_2\\
&\quad\leq\sum_{1\leq i\leq n}\;\sum_{s\in\{0,1\}^{i-1}:|s|=k}\left|\left|\widehat{B_{1\cdots i-1}}[s]\right|\right|_2\cdot\left|\left|\widehat{B_{i\cdots n}}[1\circ 0^{n-i}]\right|\right|_2\\
&\qquad\qquad\text{(by Lemma 2.1 (Decomposition))}\\
&\quad\leq(2w^2)^k\cdot 2w^2\quad\text{(by the induction hypothesis and Lemma 3.1).}
\end{aligned}$$

4 Random Restrictions

Our results involve restricting branching programs. However, our use of restrictions is different from elsewhere in the literature. Here, as in [25], we use (pseudorandom) restrictions in the usual way, but we *analyse* them by averaging over the *unrestricted* bits. Formally, we define a restriction as follows.

Definition 4.1. *For $t \in \{0,1\}^n$ and a length-n branching program B, let $B|_t$ be the* ***restriction*** *of B to t—that is, $B|_t : \{0,1\}^n \to \mathbb{R}^{w\times w}$ is a matrix-valued function given by $B|_t[x] := \underset{U}{\mathbb{E}}\left[B[\mathrm{Select}(t,x,U)]\right]$, where U is uniform on $\{0,1\}^n$.*

The most important aspect of restrictions is how they relate to the Fourier transform: For all B, s, and t, we have $\widehat{B|_t}[s] = \widehat{B}[s]$ if $s \subset t$ and $\widehat{B|_t}[s] = 0$ otherwise. The restriction t 'kills' all the Fourier coefficients that are not contained in it. This means that a restriction significantly reduces the Fourier mass:

Lemma 4.2. *Let B be a length-n, width-w program. Let T be n independent random bits each with expectation p. Then*

$$\underset{T}{\mathbb{E}}\left[L_2(B|_T)\right] = \sum_{s\neq 0} p^{|s|} \left|\left|\widehat{B}[s]\right|\right|_2 .$$

We can use Theorem 3.2 to prove a result about random restrictions of regular branching programs:

Proposition 4.3. *Let B be a length-n, width-w, read-once, oblivious, regular branching program. Let T be n independent random bits each with expectation $p \le 1/4w^2$. Then $\underset{T}{\mathbb{E}}\left[L_2(B|_T)\right] \le 1$.*

5 Pseudorandom Restrictions

To analyse our generator, we need a pseudorandom version of Proposition 4.3. That is, we need to prove that, for a *pseudorandom T* (generated using few random bits), $L_2(B|_T)$ is small. We will generate T using an almost $O(\log n)$-wise independent distribution:

Definition 5.1. *A random variable X on Ω^n is* ***δ-almost k-wise independent*** *if, for any $I = \{i_1, i_2, \cdots, i_k\} \subset [n]$ with $|I| = k$, the coordinates $(X_{i_1}, \cdots, X_{i_k}) \in \Omega^k$ are δ statistically close to being independent. We say that X is* ***k-wise independent*** *if it is 0-almost k-wise independent.*

We can sample a random variable X on $\{0,1\}^n$ that is δ-almost k-wise independent such that each bit has expectation $p = 2^{-d}$ using $O(kd + \log(1/\delta) + d\log(nd))$ random bits. See the full version of this paper for more details.

Our main lemma (stated informally as Theorem 1.3) is as follows.

Theorem 5.2 (Main Lemma). *Let B be a length-n, width-w, read-once, oblivious, regular branching program. Let T be a random variable over $\{0,1\}^n$ where each bit has expectation p and the bits are δ-almost $2k$-wise independent. Suppose $p \le (2w)^{-2}$ and $\delta \le (2w)^{-4k}$. Then*

$$\underset{T}{\mathbb{P}}\left[L_2(B|_T) \le (2w^2)^k\right] \ge 1 - n^4 \cdot \frac{2}{2^k}.$$

In particular, we show that, for $w = O(1)$, $k = O(\log n)$, and $\delta = 1/\text{poly}(n)$, we have $L_2(B|_T) \le \text{poly}(n)$ with probability $1 - 1/\text{poly}(n)$.

First we show that the Fourier mass at level $O(\log n)$ is bounded by $1/n$ with high probability. This also applies to all subprograms.

Lemma 5.3. *Let B be a length-n, width-w, ordered, regular branching program. Let T be a random variable over $\{0,1\}^n$ where each bit has expectation p and the bits are δ-almost k-wise independent. If $p \le (2w)^{-2}$ and $\delta \le (2w)^{-2k}$, then, for all $\beta > 0$,*

$$\underset{T}{\mathbb{P}}\left[\forall 1 \le i \le j \le n \quad L_2^k(B_{i\cdots j}|_T) \le \beta\right] \ge 1 - n^2 \frac{2}{2^k \beta}.$$

Proof. By Theorem 3.2, for all i and j,

$$\underset{T}{\mathbb{E}}\left[L_2^k(B_{i\cdots j}|_T)\right] = \sum_{s \subset \{i \cdots j\} : |s| = k} \underset{T}{\mathbb{P}}[s \subset T] \left|\left|\widehat{B_{i\cdots j}}[s]\right|\right|_2 \le (2w^2)^k(p^k + \delta) \le \frac{2}{2^k}.$$

The result now follows from Markov's inequality and a union bound.

Now we use Lemma 5.3 to bound the Fourier mass at higher levels. We decompose high-order ($k' \ge 2k$) Fourier coefficients into low-order ($k \le k' < 2k$) ones, similarly to the proof of Theorem 3.2:

Lemma 5.4. *Let B be a length-n, ordered branching program and $t \in \{0,1\}^n$. Suppose that, for all i, j, and k' with $1 \le i \le j \le n$ and $k \le k' < 2k$, $L_2^{k'}(B_{i\cdots j}|_t) \le 1/n$. Then, for all $k'' \ge k$ and all i and j, $L_2^{k''}(B_{i\cdots j}|_t) \le 1/n$.*

Lemmas 5.3 and 5.4 combine to give Theorem 5.2: By Lemma 5.3, a pseudorandom restriction guarantees that, with high probability the Fourier mass at levels k to $2k$ is small for all subprograms $B_{i\cdots j}$. Lemma 5.4 implies that, with high probability, the Fourier mass is small at all levels above k. The Fourier mass at levels below k can be bounded directly using Theorem 3.2.

6 The Pseudorandom Generator

Our main result (Theorem 1.1) is stated more formally as follows.

Theorem 6.1 (Main Result). *There exists a pseudorandom generator family $G_{n,w,\varepsilon} : \{0,1\}^{s_{n,w,\varepsilon}} \to \{0,1\}^n$ with seed length*

$$s_{n,w,\varepsilon} = O(w^2 \log(w) \log(n) \log(nw/\varepsilon) + w^4 \log^2(w/\varepsilon))$$

such that, for any length-n, width-w, read-once, oblivious (but unordered), permutation branching program B and $\varepsilon > 0$,

$$\left|\left|\underset{U_{s_{n,w,\varepsilon}}}{\mathbb{E}}\left[B[G_{n,w,\varepsilon}(U_{s_{n,w,\varepsilon}})]\right] - \underset{U}{\mathbb{E}}[B[U]]\right|\right|_2 \le \varepsilon.$$

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

Now we use the above results to construct our pseudorandom generator for a read-once, oblivious, permutation branching program B.

Theorem 5.2 says that with high probability over T, $B|_T$ has small Fourier mass, where T is almost k-wise independent with each bit having expectation p. This implies that $B|_T$ is fooled by small bias X and thus

$$\mathop{\mathbb{E}}_{T,X,U}\left[B[\mathrm{Select}(T,X,U)]\right] \approx \mathop{\mathbb{E}}_{T,U,U'}\left[B[\mathrm{Select}(T,U',U)]\right] = \mathop{\mathbb{E}}_{U}\left[B[U]\right].$$

If we define $\overline{B}_{t,x}[y] := B[\mathrm{Select}(t,x,y)]$, then $\mathop{\mathbb{E}}_{T,X,U}\left[\overline{B}_{T,X}[U]\right] \approx \mathop{\mathbb{E}}_{U}\left[B[U]\right]$. So now we need only construct a pseudorandom generator for $\overline{B}_{t,x}$, which is a length-$(n-|t|)$ permutation branching program. Then

$$\mathop{\mathbb{E}}_{T,X,\tilde{U}}\left[\overline{B}_{T,X}[\tilde{U}]\right] \approx \mathop{\mathbb{E}}_{T,X,U}\left[\overline{B}_{T,X}[U]\right] \approx \mathop{\mathbb{E}}_{U}\left[B[U]\right],$$

where $\tilde{U}$ is the output of the pseudorandom generator for $\overline{B}_{t,x}$. We construct $\tilde{U} \in \{0,1\}^{n-|T|}$ recursively; each time we recurse, the required output length is reduced to $n-|T| \approx n(1-p)$. Thus after $O(\log(n)/p)$ levels of recursion the required output length is constant.

The only place where the analysis breaks down for regular branching programs is when we recurse. If B is only a *regular* branching program, $\overline{B}_{t,x}$ may not be regular. However, if B is a *permutation* branching program, then $\overline{B}_{t,x}$ is too. Essentially, the only obstacle to generalising the analysis to regular branching programs is that regular branching programs are not closed under restrictions.

The pseudorandom generator is formally defined as follows.

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

```
1. Compute appropriate values of $p \in [1/8w^2, 1/4w^2]$,
   $k \geq \log_2\left(4\sqrt{w}n^4/\varepsilon\right)$, $\delta = \varepsilon(2w)^{-4k}$, and $\mu = \varepsilon(2w^2)^{-k}$.[1]
2. If $n \leq (4 \cdot \log_2(2/\varepsilon)/p)^2$, output $n$ truly random bits and stop.
3. Sample $T \in \{0,1\}^n$ where each bit has expectation $p$ and the bits
   are $\delta$-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. $\tilde{U} = G_{\lfloor n(1-p/2)\rfloor,w,\varepsilon}(U)$.
6. Sample $X \in \{0,1\}^n$ from a $\mu$-biased distribution.
7. Output $\mathrm{Select}(T,X,\tilde{U}) \in \{0,1\}^n$.
```

The analysis of the algorithm proceeds as follows.

– Every time we recurse, n is decreased to $\lfloor n(1-p/2)\rfloor$. After $O(\log(n)/p)$ recursions, n is reduced to $O(1)$ and the recursion terminates.
– The probability of failing because $|T| < pn/2$ is small by a Chernoff bound for limited independence. This requires that n is not too small (step 2).
– The output is pseudorandom, as

[1] For the purposes of the analysis we assume that p, k, δ, and μ are the same at every level of recursion. So if $G_{n,w,\varepsilon}$ is being called recursively, use the same values of p, k, δ, and μ as at the previous level of recursion.

$$\mathop{\mathbb{E}}_{T,X,\tilde{U}}\left[B[\mathrm{Select}(T,X,\tilde{U})]\right] \approx \mathop{\mathbb{E}}_{T,X,U}\left[B[\mathrm{Select}(T,X,U)]\right] \approx \mathop{\mathbb{E}}_{U}\left[B[U]\right].$$

The first approximate equality holds because we inductively assume that $\tilde{U}$ is pseudorandom; the second holds as a result of the main lemma.

– The total seed length is the seed length needed to sample X and T at each level of recursion and $O((\log(1/\varepsilon)/p)^2)$ truly random bits at the last level. Sampling X requires seed length $O(\log(n/\mu))$ and sampling T requires seed length $O(k\log(1/p) + \log(\log(n)/\delta))$.

For more details, see the full version of this paper.

7 General Read-Once, Oblivious Branching Programs

With a different setting of parameters, our pseudorandom generator can fool arbitrary oblivious, read-once branching programs, rather than just permutation branching programs (Theorem 1.2). The key to proving Theorem 1.2 is the following Fourier mass bound for arbitrary branching programs.

Lemma 7.1. *Let B be a length-n, width-w, read-once, oblivious branching program. Then, for all $k \in [n]$, $L_2^k(B) \le \sqrt{wn^k}$.*

Proof. By Parseval's Identity,

$$\sum_{s\in\{0,1\}^n:|s|=k} \left|\left|\widehat{B}[s]\right|\right|_2^2 \le \sum_{s\in\{0,1\}^n} \left|\left|\widehat{B}[s]\right|\right|_{\mathrm{Fr}}^2 = \mathop{\mathbb{E}}_{U}\left[||B[U]||_{\mathrm{Fr}}^2\right] = w.$$

The result follows from Cauchy-Schwartz.

For more details, see the full version of this paper.

References

1. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. J. ACM 53(3), 307–323 (2006)
2. Sivakumar, D.: Algorithmic derandomization via complexity theory. In: CCC 2002, p. 10 (2002)
3. Celis, L.E., Reingold, O., Segev, G., Wieder, U.: Balls and bins: Smaller hash families and faster evaluation. In: FOCS 2011, pp. 599–608 (2011)
4. Healy, A., Vadhan, S., Viola, E.: Using nondeterminism to amplify hardness. SIAM Journal on Computing 35(4), 903–931 (2006)
5. Kaplan, E., Naor, M., Reingold, O.: Derandomized constructions of k-wise (almost) independent permutations. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 354–365. Springer, Heidelberg (2005)
6. Haitner, I., Harnik, D., Reingold, O.: On the power of the randomized iterate. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 22–40. Springer, Heidelberg (2006)

7. Nisan, N.: $\mathcal{RL} \subset \mathcal{SC}$. In: STOC 1992, pp. 619–623. ACM (1992)
8. Nisan, N., Zuckerman, D.: More deterministic simulation in logspace. In: STOC 1993, pp. 235–244. ACM (1993)
9. Raz, R., Reingold, O.: On recycling the randomness of states in space bounded computation. In: STOC 1999, pp. 159–168 (1999)
10. Even, G., Goldreich, O., Luby, M., Nisan, N., Velickovic, B.: Efficient approximation of product distributions. Random Struct. Algorithms 13(1), 1–16 (1998)
11. Linial, N., Luby, M., Saks, M.E., Zuckerman, D.: Efficient construction of a small hitting set for combinatorial rectangles in high dimension. Combinatorica 17(2), 215–234 (1997)
12. Armoni, R., Saks, M.E., Wigderson, A., Zhou, S.: Discrepancy sets and pseudorandom generators for combinatorial rectangles. In: FOCS 1996, pp. 412–421 (1996)
13. Lu, C.J.: Improved pseudorandom generators for combinatorial rectangles. Combinatorica 22(3), 417–434 (2002)
14. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4), 17:1–17:24 (2008)
15. Reingold, O., Trevisan, L., Vadhan, S.: Pseudorandom walks on regular digraphs and the $\mathcal{RL}$ vs. $\mathcal{L}$ problem. In: STOC 2006, pp. 457–466. ACM (2006)
16. Saks, M., Zhou, S.: $\mathrm{BP_HSPACE}(S) \subset \mathrm{DSPACE}(S^{3/2})$. J. CSS 58(2), 376–403 (1999)
17. Bogdanov, A., Dvir, Z., Verbin, E., Yehudayoff, A.: Pseudorandomness for width 2 branching programs. ECCC TR09-070 (2009)
18. Šíma, J., Žák, S.: A sufficient condition for sets hitting the class of read-once branching programs of width 3. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 406–418. Springer, Heidelberg (2012)
19. Braverman, M., Rao, A., Raz, R., Yehudayoff, A.: Pseudorandom generators for regular branching programs. In: FOCS 2010, pp. 40–47 (2010)
20. Brody, J., Verbin, E.: The coin problem and pseudorandomness for branching programs. In: FOCS 2010, pp. 30–39 (2010)
21. Koucký, M., Nimbhorkar, P., Pudlák, P.: Pseudorandom generators for group products. In: STOC 2011, pp. 263–272. ACM (2011)
22. De, A.: Pseudorandomness for permutation and regular branching programs. In: CCC 2011, pp. 221–231 (2011)
23. Steinke, T.: Pseudorandomness for permutation branching programs without the group theory. ECCC TR12-083 (2012)
24. Impagliazzo, R., Nisan, N., Wigderson, A.: Pseudorandomness for network algorithms. In: STOC 1994, pp. 356–364 (1994)
25. Gopalan, P., Meka, R., Reingold, O., Trevisan, L., Vadhan, S.: Better pseudorandom generators from milder pseudorandom restrictions. In: FOCS 2012, pp. 120–129 (2012)
26. Naor, J., Naor, M.: Small-bias probability spaces: Efficient constructions and applications. SIAM J. Comput. 22, 838–856 (1993)
27. Tzur, Y.: Notions of weak pseudorandomness and $\mathrm{GF}(2^{\mathrm{n}})$-polynomials. Master's thesis, Weizmann Institute of Science (2009)
28. Bogdanov, A., Papakonstantinou, P.A., Wan, A.: Pseudorandomness for read-once formulas. In: FOCS 2011, pp. 240–246 (2011)
29. Impagliazzo, R., Meka, R., Zuckerman, D.: Pseudorandomness from shrinkage. In: FOCS 2012, pp. 111–119 (2012)
30. Rozenman, E., Vadhan, S.: Derandomized squaring of graphs. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX and RANDOM 2005. LNCS, vol. 3624, pp. 436–447. Springer, Heidelberg (2005)