Electronic Colloquium on Computational Complexity, Report No. 76 (2014)

Pseudorandomness and Fourier Growth Bounds for Width 3 Branching Programs

Thomas Steinke* tsteinke@seas.harvard.edu
Salil Vadhan† salil@seas.harvard.edu
Andrew Wan‡ atw12@seas.harvard.edu

27 May 2014

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 $k \in [n]$,

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

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 is tight up to a factor of $\log \log n$.

*School of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge MA. Supported by NSF grant CCF-1116616 and the Lord Rutherford Memorial Research Fellowship.

†School of Engineering and Applied Sciences, Harvard University, 33 Oxford Street, Cambridge MA. Supported in part by NSF grant CCF-1116616, US-Israel BSF grant 2010196, and a Simons Investigator Award.

‡Simons Institute for the Theory of Computing, UC Berkeley. Part of this work was completed while at Harvard University.

ISSN 1433-8092

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 [25], deterministic dimension reduction and SDP rounding [38, 16], hashing [13], hardness amplification [22], almost k-wise independent permutations [26], and cryptographic pseudorandom generator constructions [21].

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 [32] 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 [35, 5], and regular or permutation branching programs (of constant width) [9, 10, 27, 14, 40]. For width 3, hitting set generators (a relaxation of pseudorandom generators) have been constructed [42, 18]. The vast majority of these works are based on Nisan's original generator or its variants by Impagliazzo, Nisan, and Wigderson [24] and Nisan and Zuckerman [33], 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 [10]).

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 [6, 23, 34] 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

[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.

result. For unordered branching programs of length n and width w, Impagliazzo, Meka, and Zuckerman [23] 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 [6].[2] Reingold, Steinke, and Vadhan [34] 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. [18]; 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 [30]. A 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. [18] does not rely on the order in which the output bits are read, and the previously mentioned work by Reingold, Steinke, and Vadhan [34] 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 $\tilde{O}(\log^6 n)$ fools width-3 branching programs:

Theorem 1.1 (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 [23]. The construction of the generator in Theorem 1.1 is essentially the same as the generator of Gopalan et al. [18] for read-once CNF formulas, which was used by Reingold et al. [34] 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 [19]), and its study has applications to learning theory [28, 29], communication complexity [20, 1, 41, 37], and circuit complexity [8, 11, 12]. In the study of pseudorandomness, it is well-known that small-bias

[2] A generator with seed length $\tilde{O}(\sqrt{n}\log w)$ is given in [34]. The generator in [23] 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.

generators[3] with bias ε/L (which can be sampled using a seed of length $O(\log(n \cdot L/\varepsilon))$ [30, 2]) will ε-fool any function whose Fourier mass is at most L. Width-2 branching programs have Fourier mass at most $O(n)$ [5, 35] 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 [29] to obtain an improved query algorithm for polynomial-size DNF; the junta approximation results of Friedgut [17] and Bourgain [7] 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. [34] show that the generator of Gopalan et al. [18] 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 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 1.2 ([34, 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 $\Theta_k(\log^k n)$ at level k, so the bound in Theorem 1.2 does not hold for non-regular branching programs even of width 3.

The Coin Theorem of Brody and Verbin [10] 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

[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 $s \neq \emptyset$.

[4] The Tribes function (introduced by Ben-Or and Linial [4]) is 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.

product distributions captures much of the power of a class of functions, this leads to the following conjecture.

Conjecture 1.3 ([34])**.** *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 $O(\cdot)$ may depend on w.

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

Theorem 1.4 (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. [34], implies our main result (Theorem 1.1).

The Tribes function of [4] shows that the base of $O(\log n)$ of the Fourier growth in Theorem 1.4 is tight up to a factor of $\log \log n$. (See Appendix C.)

We also prove Conjecture 1.3 with $k = 1$ for any constant width w:

Theorem 1.5. *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-w, length-n, read-once, oblivious branching program. Then*

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

The proof is left to Appendix B.

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 1.2).

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 [10]: when we apply a random restriction (where each variable is kept free with probability roughly $1/k \log n$) 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 [39] (in our argument each chunk will correspond to a single layer in [39]). 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.

1.4 Organization

In Section 2 we introduce the definitions and tools we use in our proof. In Section 2.1 we formally define branching programs and explain our view of them as matrix-valued functions. In Sections 2.3 and 2.5 we define the matrix-valued Fourier transform and explain how we use it.

We prove the upper bound on Fourier mass of oblivious, read-once, width-3 branching programs (i.e., Theorem 1.4) in Section 3 (Theorem 3.1). In Sections 4 and Section 5 we construct and analyse our pseudorandom generator, which proves the main result (Theorem 1.1). The proof of Theorem 1.5 is left to Appendix 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 $B[x](u)$.

Often we think of B as having a fixed **start state** u_0 and a set $S \subset [w]$ of **accept states**. Then B **accepts** $x \in \{0,1\}^n$ if $B[x](u_0) \in S$. We say that B **computes the function** $f : \{0,1\}^n \to \{0,1\}$ if $f(x) = 1$ if and only if $B[x](u_0) \in S$.

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 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—that is,

$$\mathop{\mathbb{E}}_{X}\left[B[X]\right](u,v) = \mathop{\mathbb{P}}_{X}\left[B[X](u) = v\right],$$

where $\mathop{\mathbb{P}}_{R}[e(R)]$ is the probability of an event e with respect to the random variable R.

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.

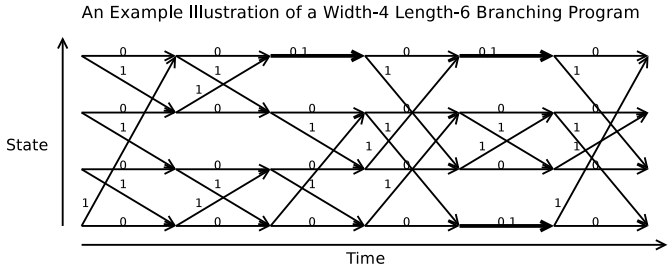
Let B and B' be width-w programs of length n and n' respectively. We define the **concatenation** $B \circ B' : \{0,1\}^{n+n'} \times [w] \to [w]$ of B and B' by

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

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** (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 $B_{i\cdots j} := B_i \circ B_{i+1} \circ \cdots \circ B_j$.

A length-n, width-w, ordered branching program can also be viewed as a directed acyclic graph. The vertices are arranged into $n+1$ layers each of size w. The edges go from one layer to the next. In particular, there is an edge labelled b from vertex u in layer $i-1$ to vertex $B[b](u)$ in layer i.



We use the following notational conventions when referring to layers of a length-n branching program. We need to distinguish between layers of vertices and layers of edges (although this is often clear from context). Layers of *edges* are the B_is and are numbered from 1 to n. Layers of *vertices* are the states between the B_is and are numbered from 0 to n. The edges in layer i (B_i) go from vertices in layer $i-1$ to vertices in layer i.

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 $\mathop{\mathbb{E}}_{X}[B[X]]$ is **stochastic**—that is,

$$\sum_{v} \mathop{\mathbb{E}}_{X}\left[B[X]\right](u,v) = 1$$

for all u and $\mathop{\mathbb{E}}_{X}[B[X]](u,v) \geq 0$ for all u and v. A program B is called a **regular program** if the matrix $\mathop{\mathbb{E}}_{U}[B[U]]$ is **doubly stochastic**—that is, both $\mathop{\mathbb{E}}_{U}[B[U]]$ and its transpose $\mathop{\mathbb{E}}_{U}[B[U]]^*$ 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 $\mathop{\mathbb{E}}_{U}[B[U]]$, whereas, if B is a permutation program, the uniform distribution is stationary for $\mathop{\mathbb{E}}_{X}[B[X]]$ for *any* distribution X.

A **regular branching program** is a branching program where each layer B_i is a regular program and likewise for a **permutation branching program**. We will refer to layer i as regular if B_i is a regular program and we say that layer i is non-regular otherwise.

Equivalently, a regular branching program is one where, in the directed acyclic graph, each vertex has in-degree 2 (in addition to having out-degree 2) except those in the start layer – that is, each layer of edges is a regular graph (hence the name). A permutation branching program has the additional constraint that the incoming edges have distinct labels.

We also 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 Norms

We are interested in constructing a random variable X (the output of the pseudorandom generator) such that $\mathop{\mathbb{E}}_{X}[B[X]] \approx \mathop{\mathbb{E}}_{U}[B[U]]$, where U is uniform on $\{0,1\}^n$. Throughout we use U to denote the **uniform distribution**. The error of the pseudorandom generator will be measured by a norm of the matrix $\mathop{\mathbb{E}}_{X}[B[X]] - \mathop{\mathbb{E}}_{U}[B[U]]$.

For a matrix $A \in \mathbb{R}^{w \times w'}$, define the ρ **operator norm** of A by

$$||A||_\rho = \max_x \frac{||xA||_\rho}{||x||_\rho},$$

where ρ specifies a vector norm (usually 1, 2, or ∞ norm). Define the **Frobenius norm** of $A \in \mathbb{R}^{w \times w'}$ by

$$||A||_{\mathrm{Fr}}^2 = \sum_{u,v} A(u,v)^2 = \mathrm{trace}(A^*A) = \sum_\lambda |\lambda|^2,$$

where A^* is the (conjugate) transpose of A and the last sum is over the singular values λ of A. Note that $||A||_2 \leq ||A||_{\mathrm{Fr}}$ for all A.

2.3 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. 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 $w \cdot w'$ entries of $B[x]$ 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 $A : \{0,1\}^n \to \mathbb{R}^{w \times w'}$ 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 $\widehat{C}[s \circ t] = \widehat{A}[s] \cdot \widehat{B}[t]$.*
- *Expectation: $\mathop{\mathbb{E}}_{X} [B[X]] = \sum_s \widehat{B}[s] \widehat{X}(s)$.*
- *Fourier Inversion for Matrices: $B[x] = \sum_s \widehat{B}[s] \chi_s(x)$.*
- *Fourier Inversion for Distributions: $\mathop{\mathbb{P}}_{X} [X = x] = \mathop{\mathbb{E}}_{U} \left[\widehat{X}(U) \chi_U(x) \right]$.*
- *Convolution for Distributions: If $Z = X \oplus Y$, then $\widehat{Z}(s) = \widehat{X}(s) \cdot \widehat{Y}(s)$.*
- *Parseval's Identity: $\sum_{s \in \{0,1\}^n} \left|\left| \widehat{B}[s] \right|\right|_{Fr}^2 = \mathop{\mathbb{E}}_{U} \left[||B[U]||_{Fr}^2 \right]$.*
- *Convolution for Matrices: If, for all $x \in \{0,1\}^n$, $C[x] = \mathop{\mathbb{E}}_{U} [A[U] \cdot B[U \oplus x]]$, then $\widehat{C}[s] = \widehat{A}[s] \cdot \widehat{B}[s]$.*

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

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

2.4 Small-Bias Distributions

The **bias** of a random variable X over $\{0,1\}^n$ is defined as

$$\text{bias}(X) := \max_{s \neq 0} |\widehat{X}(s)|.$$

A distribution is ε**-biased** if it has bias at most ε. Note that a distribution has bias 0 if and only if it is uniform. Thus a distribution with small bias is an approximation to the uniform distribution. We can sample an ε-biased distribution X on $\{0,1\}^n$ with seed length $O(\log(n/\varepsilon))$ and using space $O(\log(n/\varepsilon))$ [30, 2].

Small-bias distributions are useful pseudorandom generators: A ε-biased random variable X is indistinguishable from uniform by any linear function (a parity of a subset of the bits of X). That is, for any $s \subset [n]$, we have $\left| \mathop{\mathbb{E}}_{X} \left[\bigoplus_{i \in s} X_i \right] - 1/2 \right| \leq 2\varepsilon$. Small bias distributions are known to be good pseudorandom generators for width-2 branching programs [5], but not width-3. For example, the uniform distribution over $\{x \in \{0,1\}^n : |x| \mod 3 = 0\}$ has bias $2^{-\Theta(n)}$, but does not fool width-3, ordered, permutation branching programs.

2.5 Fourier Mass

We analyse small bias distributions as pseudorandom generators for branching programs using Fourier analysis. Intuitively, the Fourier transform of a branching program expresses that program as a linear combination of linear functions (parities), which can then be fooled using a small-bias space.

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

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

Also, define the **Fourier mass of B at level k** as

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

Note that $L(B) = \sum_{k \geq 1} L^k(B)$. 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 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, $L(B) = L(B')$ and $L^k(B) = L^k(B')$ for all k and ρ.*

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.

If $L(B)$ is small, then B is fooled by a 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*

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

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

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

3 Fourier Analysis of Width-3 Branching Programs

In this section we prove a bound on the low-order Fourier mass of width-3, read-once, oblivious branching programs. This is key to the analysis of our pseudorandom generator. Improvements to this result directly translate to improvements in our final result.

Theorem 3.1. *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-3, read-once, oblivious branching program. Then, for all $k \in [n]$,*

$$L^k(f) \leq 8n^2 \cdot (C \cdot \log_2(3n))^k = n^2 \cdot (O(\log n))^k,$$

where C is a universal constant.[5]

To prove Theorem 3.1 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

Our reduction can be stated as follows.

Proposition 3.2. *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*

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

[5] We have not optimised any constants and only show $C \leq 10^7$.

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, in Section 3.3, 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 3.2.

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,

$$B|_{\overline{g} \leftarrow x}[y] = B[\text{Select}(g, y, x)],$$

where

$$\text{Select}(g, y, x)_i = \left\{ \begin{array}{ll} y_i & i \in g \\ x_i & i \notin g \end{array} \right\}.$$

We prove Proposition 3.2 by considering a restriction $B|_{\overline{g} \leftarrow x}$ for a carefully chosen g and a random x. We show (Lemma 3.3) that is suffices to bound the Fourier growth of the restricted program $B|_{\overline{g} \leftarrow x}$ and (Lemma 3.4) that the restricted program is of the desired form $D^{6(\ell+1)k}$ with high probability.

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 [39]. Lemma 3.3 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 $O(k)$ non-regular layers.

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

$$g_t := \bigcup_{(i \bmod m)+1 \in t} c_i$$

and

$$G_t^k := \{s \subset g_t : |s| = k\}.$$

We refer to g_t as the $t^{\textbf{th}}$ **group of indices** and G_t^k as the $t^{\textbf{th}}$ **group of (order** k**) Fourier coefficients**. The following 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 3.3. *Let B be a length-n 3OBP, $k \in [n]$, $m \geq k$ and g_t as above. Then*

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

Proof. Note that some Fourier coefficients appear in multiple G_ts, but every coefficient at level k appears in at least one G_t. Thus

$$\begin{aligned}
L^k(B) &\leq \sum_{t:|t|=k} \sum_{s \in G_t^k} \left|\left| \widehat{B}[s] \right|\right|_2 \\
&= \sum_{t:|t|=k} \sum_{s \in G_t^k} \left|\left| \mathop{\mathbb{E}}_{U} \left[\widehat{B_{\overline{g_t} \leftarrow U}}[s] \right] \right|\right|_2 \\
&\leq \sum_{t:|t|=k} \sum_{s \in G_t^k} \mathop{\mathbb{E}}_{U} \left[\left|\left| \widehat{B_{\overline{g_t} \leftarrow U}}[s] \right|\right|_2 \right] \\
&= \sum_{t:|t|=k} \mathop{\mathbb{E}}_{U} \left[L^k(B|_{\overline{g_t} \leftarrow U}) \right] \\
&\leq \binom{m}{k} \max_{t \subset [m]:|t|=k} \mathop{\mathbb{E}}_{U} \left[L^k(B|_{\overline{g_t} \leftarrow U}) \right],
\end{aligned}$$

where the second inequality follows from the convexity of the norm. □

Given Lemma 3.3, we may now prove Proposition 3.2 by giving an upper bound on $\mathop{\mathbb{E}}_{U} \left[L^k(B|_{\overline{g_t} \leftarrow U}) \right]$ for any 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 3.4. *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 $1 - n \cdot 2^{-\ell \cdot (m-k)}$ 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.

Proof. Let $t = \{t_1, t_2, \cdots, t_k\}$ where $t_1 < t_2 < \cdots < t_k$. Define $t_{a+k} = t_a + m$ for all a. Let a' be the largest value of a such that $t_a \leq n$. We redefine $t_{a'+1} = n + 1$. We can write

$$B|_{\overline{g_t} \leftarrow x} = C'_{t_1} \circ C'_{t_2} \circ \cdots \circ C'_{t_{a'}},$$

where each C'_{t_a} corresponds to C_{t_a}. However, the chunks $C_{t_a+1}, C_{t_a+2}, \cdots C_{t_{a+1}-1}$ have been restricted and C'_{t_a} reflects that. Formally, for all $a \in [l'] \backslash \{1\}$ and $y \in \{0,1\}^{|c_{t_a}|}$, we have

$$C'_{t_a}[y] = C_{t_a}[y] \cdot C_{t_a+1}[x_{c_{t_a+1}}] \cdot \cdots \cdot C_{t_{a+1}-1}[x_{c_{t_{a+1}-1}}],$$

and, for all $y \in \{0,1\}^{|c_{t_1}|}$, we have

$$C'_{t_1}[y] = C_1[x_{c_1}] \cdot \cdots \cdot C_{t_1-1}[x_{c_{t_1-1}}] \cdot C_{t_1}[y] \cdot C_{t_1+1}[x_{c_{t_1+1}}] \cdot \cdots \cdot C_{t_2-1}[x_{c_{t_2-1}}],$$

where x_{c_t} is the coordinates of x contained in c_t. Moreover, we remove any unreachable vertices. That is, if $\mathop{\mathbb{E}}_{U}\left[C'_{t_a}[U]\right]$ has a column of zeros, we remove that column (making the matrix non-square) and remove the corresponding row from $C'_{t_{a+1}}$. This reduces the width of the layer of vertices between C'_{t_a} and $C'_{t_{a+1}}$.

It should be clear that each C'_{t_a} is a 3OBP with between one and three non-regular layers. (One non-regular layer comes from C_{t_a} and the first and last layers may become non-regular after the restriction.)

Consider $\ell \cdot k + 1$ consecutive C'_{t_a}s in the restricted program $C'_{t_a} \circ C'_{t_{a+1}} \circ \cdots \circ C'_{t_{a+\ell\cdot k}}$ and the corresponding subprogram before restriction $C_{t_a} \circ \cdots \circ C_{t_{a+\ell\cdot k}}$ which contains $\ell \cdot m + 1$ chunks (recall that $t_{a+k} = t_a + m$). There are exactly $\ell \cdot k + 1$ chunks $C'_{t_a}, \ldots, C'_{t_a+k}, \ldots, C'_{t_{a+\ell\cdot k}}$ which remain free after the restriction, i.e., $\ell(m-k)$ chunks have been restricted. Furthermore, each such chunk contains a non-regular layer.

Each non-regular layer that is restricted has at least a $1/2$ probability of reducing the width: Being non-regular implies that there are two edges with the same label going to the same vertex. There is a $1/2$ probability of the restriction picking that label. If this happens, then one of the vertices on the right side of the layer becomes unreachable and therefore the width is reduced. This is the same argument that is used by Brody and Verbin [10] and Steinberger [39].

So, with probability at least $1 - 2^{-\ell(m-k)}$ over the choice of x, there exists a layer between C'_{t_a} and $C'_{t_{a+\ell\cdot k}}$ that has width at most 2. Call such a layer of vertices a **bottleneck**.

By a union bound, with probability at least $1 - n \cdot 2^{-\ell(m-k)}$ over the choice of x, every $\ell \cdot k + 1$ successive C'_{t_a}s have one such bottleneck. We split the C'_{t_a}s into groups that are separated by bottlenecks to write $B|_{\overline{g_t}\leftarrow x} = D_1 \circ D_2 \circ \cdots \circ D_r$ (one D_i per group). Since each remaining chunk has at most 3 non-regular layers, and there can be at most $2\ell k$ chunks in each group, each D_i is a 3OBP with at most $6\ell k$ non-regular layers and the layer between D_i and D_{i+1} has width at most 2 (i.e., is a bottleneck). □

We may now complete the proof of Proposition 3.2.

Proof of Proposition 3.2. By Lemma 3.3, it suffices to bound, for every fixed t, the quantity $\mathop{\mathbb{E}}_{U}\left[L^k(B|_{\overline{g_t}\leftarrow U})\right]$.

We compute the expectation of $L^k(B|_{\overline{g_t}\leftarrow U})$ by conditioning on the how far apart bottlenecks are in the restricted program and applying Lemma 3.4. Let β_x be the largest number of non-regular layers occuring in $B|_{\overline{g_t}\leftarrow x}$ that are not separated by a bottleneck (i.e., a width-2 layer).

$$\begin{aligned}\mathop{\mathbb{E}}_{U}\left[L^k(B|_{\overline{g_t}\leftarrow U})\right] &\le \sum_{\ell\ge 0} \mathop{\mathbb{P}}_{x}\left[6\ell k < \beta_x \le 6(\ell+1)k\right] \cdot \mathop{\mathbb{E}}_{x}\left[L^k(B|_{\overline{g_t}\leftarrow x}) \mid \beta_x \le 6(\ell+1)k\right] \\ &\le \sum_{\ell\ge 0} n \cdot 2^{-\ell\cdot(m-k)} \cdot L^k(D^{6(\ell+1)k})\end{aligned}$$

where $D^{6(\ell+1)k}$ is the branching program that maximizes $L^k(B|_{\overline{g_t}\leftarrow x})$ over x such that $\beta_x \le 6(\ell+1)k$. □

3.2 Part 2 – Mixing in Width-2

Now it remains to bound the Fourier mass of 3OBPs of the form given by Proposition 3.2.

Proposition 3.5. *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 $L^k(D^\ell) \le 2n \cdot (6000(\ell+1))^k = n \cdot O(\ell)^k$ for all $k, \ell \ge 1$.

Before we prove Proposition 3.5, we show how it implies a bound on Fourier mass of general 3OBPs.

Proposition 3.6. *Let B be a length-n, width-3, read-once, oblivious branching program with width 2 in the first and last layers. Then, for all $k \in [n]$,*

$$L^k(B) := \sum_{s \in \{0,1\}^n : |s| = k} |\widehat{B}[s]| \le 8n^2 \cdot (200000k)^k = n^2 \cdot O(k)^k.$$

Proof. Let B be a 3OBP computing f and assume B has width 2 in the first and last layers. By Propositions 3.2 and 3.5, we have, setting $m = 2k+1$,

$$\begin{aligned}
L^k(B) \le& n \cdot \binom{m}{k} \sum_{\ell \ge 0} 2^{-\ell(m-k)} L^k(D^{6(\ell+1)k}) \\
\le& n \cdot \binom{m}{k} \sum_{\ell \ge 0} 2^{-\ell(m-k)} 2n \cdot (6000(6(\ell+1)k+1))^k \\
\le& 2n^2 \binom{2k+1}{k} \sum_{\ell \ge 0} 2^{-\ell(k+1)} (6000(6(\ell+1)k+1))^k \\
\le& 4n^2 4^k \sum_{\ell \ge 0} 2^{-\ell} \cdot \left(6000 \frac{6(\ell+1)k+1}{2^\ell}\right)^k \\
\le& 4n^2 4^k \left(\sum_{\ell \ge 0} 2^{-\ell}\right) \cdot (6000 \cdot 7k)^k \\
\le& 8n^2 \cdot (6000 \cdot 4 \cdot 7k)^k,
\end{aligned}$$

as required. □

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,

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

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

$$\left|\left|\widehat{D}[s]\right|\right|_2 \leq \left|\left|\widehat{D_1}[s_1]\right|\right|_2 \cdot \lambda(D_2). \tag{1}$$

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 $1/(1-C)^{O(k)}$ bound on the level-k Fourier mass. We show that any D_i in the branching program of the form given by Proposition 3.2 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 3.7. *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 3.7 implies Proposition 3.5:

Proof of Proposition 3.5. 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 2.1), we have

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

The second inequality follows from Equation 1 and the third from Lemma 3.7. 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 3.7. 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 [34]. 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

We consider the case where $\lambda(D) < 0.99$. We use the following result as a black box.

Lemma 3.8 ([9, Lemma 4], [34, Lemma 3.1])**.** *Let B be a length-n, width-w, ordered, regular branching program. Then*

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

The quantity $\left|\left|\widehat{B_{i\cdots n}}[1 \circ 0^{n-i}]\right|\right|_2$ measures the correlation between the i^{th} input bit and the final state of the program, which we call the **weight** of bit i. The entry in the u^{th} row and v^{th} column of $2\widehat{B_{i\cdots n}}[1 \circ 0^{n-i}]$ is the the probability of reaching vertex v in layer n given that we reached vertex u in layer $i-1$ and the i^{th} input bit is 0 minus the same probability given that the i^{th} input bit is 1. Braverman et al. [9] proved this result for a different measure of weight. Their result was translated into the above Fourier-analytic form by Reingold et al. [34].

We can add some non-regular layers to get the following.

Lemma 3.9. *Let B be a length-n, width-w, ordered branching program with at most k non-regular layers. Then*

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

Proof. The proof proceeds by induction on k. If $k=0$, the result follows from Lemma 3.8. Suppose the result holds for some k and let B be a length-n, width-w ordered branching program with $k+1$ non-regular layers. Let i^* be the index of the first non-regular layer. Then

$$\begin{aligned}
\sum_{1\le i\le n}\left|\left|\widehat{B_{i\cdots n}[1\circ 0^{n-i}]}\right|\right|_2 &= \sum_{1\le i<i^*}\left|\left|\widehat{B_{i\cdots(i^*-1)}[1\circ 0^{i^*-i-1}]\cdot B_{i^*\cdots n}[0]}\right|\right|_2 \\
&\quad + \left|\left|\widehat{B_{i^*\cdots n}[1\circ 0^{n-i^*}]}\right|\right|_2 \\
&\quad + \sum_{i^*<i\le n}\left|\left|\widehat{B_{i\cdots n}[1\circ 0^{n-i}]}\right|\right|_2 \\
&\le \left(\sum_{1\le i<i^*}\left|\left|\widehat{B_{i\cdots(i^*-1)}[1\circ 0^{i^*-i-1}]}\right|\right|_2\right)\cdot\left|\left|\widehat{B_{i^*\cdots n}[0]}\right|\right|_2 \\
&\quad + \sqrt{w} + (2w^2+1)\sqrt{w}(k+1) \\
&\le 2w^2\cdot\sqrt{w}+\sqrt{w}+(2w^2+1)\sqrt{w}(k+1) \\
&\le (2w^2+1)\sqrt{w}(k+2),
\end{aligned}$$

where we use the fact that $\left|\left|\widehat{B}[s]\right|\right|_2 \le \sqrt{w}$ for any s and width-w branching program B. □

This gives us the following bound on the Fourier mass.

Theorem 3.10. *Let B be a length-n, width-w, orderd branching program with at most k non-regular layers. Then, for all $k'\in[n]$,*

$$L^{k'}(B) := \sum_{s\in\{0,1\}^n:|s|=k}\left|\left|\widehat{B}[s]\right|\right|_2 \le \sqrt{w}\cdot((2w^2+1)\sqrt{w}(k+1))^{k'} \le \sqrt{w}\cdot(3w^{2.5}(k+1))^{k'}.$$

This result is proved using Lemma 3.9 analogously to how [34, Theorem 3.2] is proved using Lemma 3.8.

Proof. We perform an induction on k'. If $k'=0$, then there is only one Fourier coefficient to bound—namely, $\widehat{B}[0^n] = \mathop{\mathbb{E}}_U[B[U]]$. Since $\mathop{\mathbb{E}}_U[B[U]]$ is stochastic, $\left|\left|\mathop{\mathbb{E}}_U[B[U]]\right|\right|_2 \le \sqrt{w}$ and the base case follows. Now suppose the bound holds for k' and consider $k'+1$. We split the Fourier

coefficients based on where the last 1 is:

$$\begin{aligned}
&\sum_{s\in\{0,1\}^n:|s|=k'+1} \left|\left|\widehat{B}[s]\right|\right|_2 \\
&= \sum_{1\le i\le n} \sum_{s\in\{0,1\}^{i-1}:|s|=k'} \left|\left|\widehat{B}[s\circ 1\circ 0^{n-i}]\right|\right|_2 \\
&= \sum_{1\le i\le n} \sum_{s\in\{0,1\}^{i-1}:|s|=k'} \left|\left|\widehat{B_{1\cdots i-1}}[s]\cdot\widehat{B_{i\cdots n}}[1\circ 0^{n-i}]\right|\right|_2 \quad \text{(by Lemma 2.1 (Decomposition))} \\
&\le \sum_{1\le i\le n} \sum_{s\in\{0,1\}^{i-1}:|s|=k'} \left|\left|\widehat{B_{1\cdots i-1}}[s]\right|\right|_2 \cdot \left|\left|\widehat{B_{i\cdots n}}[1\circ 0^{n-i}]\right|\right|_2 \\
&\le \sum_{1\le i\le n} \sqrt{w}\cdot((2w^2+1)\sqrt{w}(k+1))^{k'} \cdot \left|\left|\widehat{B_{i\cdots n}}[1\circ 0^{n-i}]\right|\right|_2 \quad \text{(by the induction hypothesis)} \\
&\le \sqrt{w}\cdot((2w^2+1)\sqrt{w}(k+1))^{k'}\cdot(2w^2+1)\sqrt{w}(k+1) \quad \text{(by Lemma 3.9)} \\
&= \sqrt{w}\cdot((2w^2+1)\sqrt{w}(k+1))^{k'+1},
\end{aligned}$$

as required. □

Lemma 3.11. *Let D be a 3OBP with at most k non-regular layers. If $p \le 1/6000(k+1)$, then $L_p(D) \le 1/100$.*

Proof. We have

$$\begin{aligned}
L_p(D) &= \sum_{k'\ge 1} p^{k'} L^{k'}(D) \\
&\le \sum_{k'\ge 1} p^{k'}\sqrt{3}((2\cdot 3^2+1)\sqrt{3}(k+1))^{k'} \\
&\le \sqrt{3}\sum_{k'\ge 1}\left(\frac{19\sqrt{3}(k+1)}{6000(k+1)}\right)^{k'} \\
&\le 1/100.
\end{aligned}$$

□

It immediately follows that $\lambda(D) + L_p(D) \le 1$ when $p \le 1/6000(k+1)$, assuming $\lambda(D) < 0.99$. This covers the 'easy' case of Lemma 3.7.

Hard Case – Poor Mixing

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

Lemma 3.12. *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 3.7 and, along with Lemma 3.11 completes the proof of Lemma 3.7.

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, 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

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

   as required.

The same holds for g, which gives the result.

Step 1.

Lemma 3.13. *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.*

Proof. We assign each vertex of D a **charge**: The two vertices v_+ and v_- in the first layer are assigned charges 1 and -1 respectively. Each edge is assigned a charge that is half the charge of the vertex it originates from and the charge of each subsequent vertex is the sum of the charges of the incoming edges. In other words, the charge of a vertex u is the probabililty that a random walk from v_+ reaches u minus the probability that a random walk from v_- reaches u.

The partition is given by the sign of the charge: Let Q be the set of vertices with positive or zero charge and let $\overline{Q}$ be the set of vertices with negative charge. Now we must prove that there are $O(k)$ edges crossing between Q and $\overline{Q}$.

Define the **total charge** of a layer to be the sum of the absolute values of the charges in that layer. Clearly the total charge cannot increase from one layer to the next (by the triangle inequality). Moreover, the total charge in the final layer equals $((1-\alpha)-\alpha)+((1-\beta)-\beta) = 2\lambda(D)$.

By assumption ($\lambda(D) \geq 0.99$) the total charge decreases by at most $1/50$. The total charge only decreases when an edge crosses the $(Q, \overline{Q})$ partition, as this is when positive and negative charges cancel. In fact, it decreases by precisely the charge of the crossing edge.

So there is very little charge crossing the partition. However, it is possible that many edges with little charge cross the partition. To preclude this possibility, we also track the **minimum charge** of each layer, which is the minimum absolute value of a charge of a vertex in that layer.

Now we use the fact that there are at most k non-regular layers in D. Call a layer a **crossing layer** if it contains an edge that crosses the partition i.e. where the signs of the charges of the endpoints of the edge are different. Clearly there are at most k non-regular crossing layers. We need only account for regular crossing layers.

Consider a regular crossing layer. We will argue that the minimum charge must go from 'small' to 'large'. Then we will argue that in order for the minimum charge to go from large back to small, a non-regular layer is needed. So each such regular crossing layer must have a corresponding non-regular layer. This ensures that there are at most $k+1$ regular crossing layers, as required.

Let B_i be a regular crossing layer. Let $a \leq b \leq c$ be the charges on the left vertices of the layer. Since $a+b+c = 0$ and $|a|+|b|+|c| \geq 1.98$, we have that $a \leq -0.49$ and $c \geq 0.49$. The vertices corresponding to a and c cannot have a common neighbour, as otherwise the total charge would decrease by at least 0.2. Up to permuting vertices, this leaves three possibilities for the layer, which we depict in Figure 1.

Possibility (iii) does not have a crossing, so can be ignored. Possibilities (i) and (ii) are essentially the same up to flipping signs. So let's analyse possibility (i).

For there to be a crossing, we must have $b < 0$. The total charge then decreases by $|b|$. So $|b| \leq 1/50$. Now $|b|$ is the minimum charge of the vertices on the left. So the minimum charge

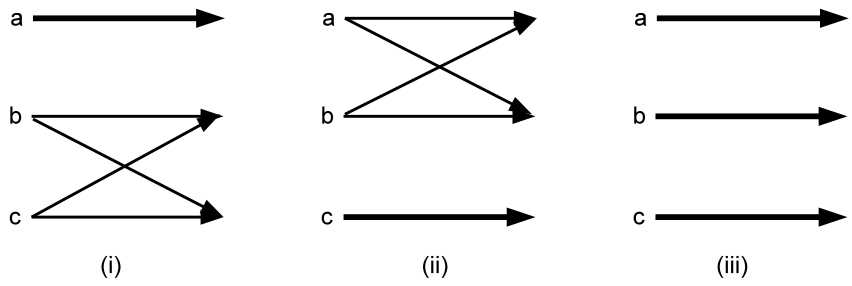


Figure 1: The bold arrows indicate double edges.

on the left of a regular crossing layer is at most $1/50$. On the right, the minimum charge is $\min\{|a|, |b+c|/2\} \geq \min\{0.49, (0.49-1/50)/2\} > 1/5$. So this layer increases the minimum charge by at least $1/5 - 1/50 > 0.1$.

Now we will show that any two regular crossing layers must have a non-regular layer between them. For the sake of contradiction, let B_i and B_j $(i < j)$ be two regular crossing layers with no non-regular layers between them. We can assume that there are no crossing layers between B_i and B_j: If not, replace B_j with the first crossing layer after B_i.

Now consider the vertices in B_{i+1}, call them v_1, v_2 and 3. We assume without loss of generality that one vertex has positive charge (say v_1), while v_2 and v_3 have negative charge. Because there are no crossing layers, no path from v_1 can share a vertex with any path starting from v_2 or v_3. Thus, up to permutation on the vertices (determining which vertex is "isolated"), the first column and first row of the matrix $\mathop{\mathbb{E}}_{U}\left[B_{i+1,\ldots,j-1[U]}\right]$ are equal to $(1,0,0)$. Because every layer of $B_{i+1,\ldots,j-1}$ must be regular, up to permuting vertices, we have that $\mathop{\mathbb{E}}_{U}[B_{i+1\cdots j-1}[U]]$ is of the form

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix}.$$

The first possibility cannot decrease the minimum charge at all. The second possibility can only decrease the minimum charge by cancellation: Let a, b, and c be the charges on the left. Then the charges on the right are a, $(b+c)/2$, and $(b+c)/2$. The only way the minimum charge can decrease is if b and c have opposite signs. By symmetry, we can assume that $c \geq -b \geq 0$. Thus the new minimum charge is either $|a|$ or $(c-|b|)/2 = (c+|b|)/2 - |b|$. So the minimum charge decreases by at most $|b|$. However, the total charge decreases by $|b| + |c| - |b+c| = c + |b| - (c-|b|) = 2|b|$. Since the total charge can decrease by at most $1/50$, we have $|b| \leq 1/100$ and the minimum charge can decrease by at most $1/100$ – a contradiction, as it must decrease from at least $1/5$ to at most $1/50$.

Thus we have shown that each pair of regular crossing layers has a non-regular layer between then. Since there are at most k non-regular layers, there are at most $2k+1$ crossing layers, as required. $\square$

Step 2.

Lemma 3.14. *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.*

Proof. Call a layer of edges of D **critical** if it is either non-regular or it has an edge crossing the partition given by Lemma 3.13. Let Γ be the set of critical layers. By Lemma 3.13, D has at most $3k+1$ critical layers. Between critical layers, D is partitioned into two width-2 regular branching programs. (To be more precise, it is partitioned into a width-2 regular branching program and a width-1 regular branching program.)

Define $\tilde{\Gamma}$ to be the set of 'fixings' of edges in Γ, that is, $s \in \tilde{\Gamma}$ specifies for each $i \in \Gamma$ an edge $s(i)$ in layer i (specifiing one of three states to the left of layer i and the label, which is 0 or 1, of the edge taken). We can think of $s \in \tilde{\Gamma}$ as a function $s : \Gamma \to [3] \times \{0,1\}$.

For $s \in \tilde{\Gamma}$, define $f_s : \{0,1\}^d \to \{0,1\}$ to be the following indicator function:

$$f_s(x) = 1 \iff f(x) = 1 \wedge \text{the path in } D \text{ given by } x \text{ uses all the edges in } s.$$

Clearly $f(x) = \sum_{s \in \tilde{\Gamma}} f_s(x)$, as each path in D is consistent with exactly one s.

Now we claim that each f_s can be written as the conjunction of at most $2|\Gamma|+1$ regular width-2 branching programs. In particular, there is one term for each critical layer and one term for each gap between critical layers.

Let $i_1 < i_2 < \cdots < i_{|\Gamma|}$ be an enumeration of Γ. (Also define $i_0 = 0$ and $i_{|\Gamma|+1} = d+1$.) We will write

$$f_s(x) = \prod_{j=1}^{2|\Gamma|+1} f_{s,j}(x_j),$$

where the x_js are a partition of x as follows. For $j \in [|\Gamma|+1]$, $x_{2j-1} \in \{0,1\}^{i_j - i_{j-1} - 1}$ contains the coordinates from $i_{j-1}+1$ to $i_j - 1$ of x. For $j \in [|\Gamma|]$, $x_{2j} \in \{0,1\}$ is coordinate i_j of x. The function $f_{s,j}$ checks the bits in x_j and is 1 if and only if the path is consistent with $f_s = 1$ (assuming the bits outside of x_j are set consistently with $f_s = 1$. Thus $f_{s,2j-1}(x_{2j-1})$ verifies that if started in the state $s(i_{j-1})_1$, the input x_{2j-1} leads D to state $s(i_j)_1$ in layer i_j, and $f_{s,2j}(x_{2j})$ verifies that $x_{2j} = s(i_{2j})_2$, i.e., that the setting of x_{2j} is the same as the label specified by $s(i_{2j})$. Note that for $f_{s,j}$ to be satisfied, there is only one correct vertex at each end of the path. The functions $f_{s,2j}$ are determined by a single literal and can be computed by width-2 branching programs, and since the functions $f_{s,2j-1}$ are computed over non-critical layers from a single starting vertex, they are also computed by width-2. □

Step 3.

We use the following fact about regular width-2 ordered branching programs – a very simple class of functions.

Lemma 3.15. *Let $f : \{0,1\}^n \to \{0,1\}$ be computed by a width-2 regular ordered branching program. Then $\mathop{\mathbb{E}}_U[f(U)] \in \{0, 1/2, 1\}$ and $L_p(f) \le p/2$ for all $p \in [0,1]$.*

Proof. Every layer of a regular width-2 ordered branching program falls into one of three cases: trivial layers (the input bit does not affect the state), a (negated) XOR (flips the state depending on the input bit), or a (negated) dictator (sets the state based on the current input bit regardless of the previous state). Thus any such branching program is either a constant function, which gives $\mathop{\mathbb{E}}_U[f(U)] \in \{0,1\}$ and $L_p(f) = 0$, or is a (possibly negated) XOR of a subset of the input bits. In the latter case f has one non-trivial coefficient of magnitude $1/2$, which implies $\mathop{\mathbb{E}}_U[f(U)] = 1/2$ and $L_p(f) \le p \cdot L(f) \le p/2$. □

Lemma 3.16. *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 $L_p(f) \le 2kp \cdot \mathop{\mathbb{E}}_U[f(U)]$ for any $p \le 1/k$.*

Proof. Define $\alpha_j = \mathop{\mathbb{E}}_U[f_j(U)]$. We have $\alpha = \prod_{j\in[k]} \alpha_j$. Now

$$\begin{aligned}
\alpha + L_p(f) &= \sum_s p^{|s|} |\widehat{f}[s]| \\
&= \sum_s p^{|s|} \prod_{j\in[k]} |\widehat{f_j}[s_j]| \\
&= \prod_{j\in[k]} \sum_{s_j} p^{|s_j|} |\widehat{f_j}[s_j]| \\
&= \prod_{j\in[k]} (\alpha_j + L_p(f_j)) .
\end{aligned}$$

Since f_j is computed by a width-2 regular branching program, $\alpha_j \in \{0, 1/2, 1\}$. If $\alpha_j = 0$, then $\alpha = 0$ and $L_p(f) = 0$, so we can ignore this case. Moreover, if $\alpha_j = 1$, then $f_j = 1$ is constant and $L_p(f_j) = 0$, so we can ignore the terms with $\alpha_j = 1$. Let $J = \{j \in [k] : \alpha_j = 1/2\}$. Thus we are left with

$$\begin{aligned}
L_p(f) =& \prod_{j\in J} \left(\frac{1}{2} + L_p(f_j)\right) - 2^{-|J|} \\
=& \alpha \cdot \left(\prod_{j\in J} (1 + 2L_p(f_j)) - 1\right) \\
\le& \alpha \cdot \left(\prod_{j\in J} (1 + p) - 1\right) \\
\le& \alpha \cdot \left((e^p)^{|J|} - 1\right) \\
\le& \alpha \cdot 2p|J| \\
\le& \alpha 2pk.
\end{aligned}$$

as long as $p|J| \le 1$. □

Proof of Lemma 3.12. By Lemma 3.14, we can write $f(x) = \sum_s \prod_j f_{s,j}(x_j)$. Let $f_s(x) = \prod_j f_{s,j}(x_j)$, where the product is over at most $6k+3$ terms. Then, by Lemma 3.16,

$$L_p(f) \le \sum_s L_p(f_s) \le \sum_s (12k+6)p \cdot \mathop{\mathbb{E}}_U [f_s(U)] = (12k+6)p\alpha,$$

as long as $p \le 1/(6k+3)$. Likewise $L_p(g) \le (12k+6)p\beta$ for $p \le 1/(6k+3)$.

Now $L_p(D) \le 2L_p(f) + 2L_p(g) \le (24k+12) \cdot p \cdot (\alpha+\beta)$. If $p \le 1/(24k+12)$, then $L_p(D) + \lambda(D) \le \alpha + \beta + 1 - \alpha - \beta = 1$, as required. □

3.3 Bootstrapping

Proposition 3.6 gives a bound on the Fourier growth of width-3 branching programs of the form $L^k(B) \le n^2 \cdot O(k)^k$. The k^k term is inconvenient, but can be easily removed by "bootstrapping":

The following proposition shows that if we can bound the Fourier mass up to level $O(\log n)$, then we can bound the Fourier mass at all levels.

Proposition 3.17. *Let B be a length-n, ordered branching program such that, for all $i, j, k \in [n]$ with $k \le 2k^*$ and $i \le j$, we have $L^k(B_{i\cdots j}) \le a \cdot b^k$. Suppose $a \cdot n \le 2^{k^*}$. Then, for all $i, j, k \in [n]$ with $i \le j$, we have $L^k(B_{i\cdots j}) \le a \cdot (2b)^k$.*

The proof is similar to that of Lemma 4.4.

Proof. Suppose the proposition is false and fix the smallest k such that the statement does not hold. Clearly $k > 2k^*$. Let $k' = k - k^*$. By minimality $L^{k'}(B_{i\cdots j}) \le a \cdot (2b)^{k'}$ for all $i \le j$. Now

$$L^k(B_{i\cdots j}) \le \sum_{\ell=i}^{j+1} L^{k'}(B_{i\cdots \ell-1}) \cdot L^{k^*}(B_{\ell\cdots j}) \le \sum_{\ell=i}^{j+1} a \cdot (2b)^{k'} \cdot a \cdot b^{k^*} \le n \cdot a \cdot (2b)^{k'} \cdot a \cdot b^{k^*} = a \cdot (2b)^k \cdot \left(\frac{na}{2^{k^*}}\right).$$

Since $na \le 2^{k^*}$, we have a contradiction, as we assumed $L^{k^*}(B_{\ell\cdots j}) > a \cdot (2b)^k$. □

Now we combine Propositions 3.6 with 3.17 to prove Theorem 3.1

Proof of Theorem 3.1. Let B be a length-n 3OBP computing f with width 2 in the first and last layers. By Proposition 3.6, we have

$$L^k(B) \le 8n^2 \cdot (200000k)^k$$

for any length-n 3OBP B and $k \in [n]$. Since a subprogram of a 3OBP is also a 3OBP, this bound also applies to $L^k(B_{i\cdots j})$ for all $i, j \in [n]$. If we set $k^* = \lceil \log_2(8n^3) \rceil$, $a = 8n^2$, and $b = 200000 \cdot 2k^*$, then the hypotheses of Proposition 3.17 are satisfied. Thus, for any $k \in [n]$, we have

$$L^k(B) \le 8n^2 \cdot (2 \cdot 200000 \cdot 2k^*)^k.$$

Since $L^k(f) \le L^k(B)$, this gives the result. □

4 Pseudorandom Restrictions

Our pseudorandom generator repeatedly applys pseudorandom restrictions. For the analysis, we introduce the concept of an averaging restriction as in Gopalan et al. [18] and Reingold et al. [34], which is subtly different to the restrictions in Section 3.

Definition 4.1. *For $t \in \{0,1\}^n$ and a length-n branching program B, let $B|_t$ be the **(averaging) 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 $B|_t[x] := \mathop{\mathbb{E}}_U [B[\mathrm{Select}(t, x, U)]]$, where U is uniform on $\{0,1\}^n$.*

In this section we show that, for a *pseudorandom* T (generated using few random bits), $L(B|_T)$ is small. We will generate T using an almost $O(\log n)$-wise independent distribution:

Definition 4.2. *A random variable X on Ω^n is δ-**almost k-wise independent** if, for every $I = \{i_1, i_2, \cdots, i_k\} \subset [n]$ with $|I| = k$, the coordinates $(X_{i_1}, X_{i_2}, \cdots, X_{i_k}) \in \Omega^k$ are δ-close (in statistical distance) to being independent—that is, for all $T \subset \Omega^k$,*

$$\left| \sum_{x \in T} \left(\mathop{\mathbb{P}}_X [(X_{i_1}, X_{i_2}, \cdots, X_{i_k}) = x] - \prod_{l \in [k]} \mathop{\mathbb{P}}_X [X_{i_l} = x_l] \right) \right| \leq \delta.$$

*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 [34, Lemma B.2].

The following lemma, proven in essentially the same way as Lemma 5.3 in [34], tells us that $L(B|_T)$ will be small for T chosen from a δ-almost k-wise distribution with appropriate parameters.

Lemma 4.3. *Let B be a length-n, width-w, ordered 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 that, for all $i, j, k' \in [n]$ such that $k \leq k' < 2k$, we have $L^{k'}(B_{i \cdots j}) \leq a \cdot b^{k'}$. If we set $p \leq 1/2b$ and $\delta \leq 1/(2b)^{2k}$, then*

$$\mathop{\mathbb{P}}_T \left[L^{\geq k}(B|_T) > 1 \right] \leq n^4 \cdot \frac{2a}{2^k}.$$

(Recall that $L^{\geq k}(g) = \sum_{j=k}^{n} L^j(g)$.)

Proof. Let $k \leq k' < 2k$. We have that for all i and j,

$$\mathop{\mathbb{E}}_T \left[L^{k'}(B_{i \cdots j}|_T) \right] = \sum_{s \subset \{i \cdots j\} : |s| = k'} \mathop{\mathbb{P}}_T [s \subset T] \left\| \widehat{B_{i \cdots j}}[s] \right\|_2 \leq L^{k'}(B)(p^{k'} + \delta) \leq a b^{k'} \left(\frac{1}{(2b)^{k'}} + \frac{1}{(2b)^{2k}} \right) \leq \frac{2a}{2^k}.$$

Applying Markov's inequality and a union bound, we have that for all $\beta > 0$:

$$\mathop{\mathbb{P}}_T \left[\forall 1 \leq i \leq j \leq n \;\; L_2^{k'}(B_{i \cdots j}|_T) \leq \beta \right] \geq 1 - n^2 \frac{2a}{2^k \beta}.$$

Applying a union bound over values of k' and setting $\beta = 1/n$, we obtain:

$$\mathop{\mathbb{P}}_{T}\left[\forall k \le k' < 2k\ \forall 1 \le i \le j \le n \;\; L^{k'}(B_{i\cdots j}|_T) \le \frac{1}{n}\right] \ge 1 - n^4 \cdot \frac{2a}{2^k}.$$

The result now follows from the following Lemma.

Lemma 4.4 ([34, 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, $L_2^{k''}(B_{i\cdots j}|_t) \le 1/n$.*

□

5 The Pseudorandom Generator

Our main result Theorem 1.1 follows from plugging our Fourier growth bound (Theorem 3.1) into the analysis of [34]. We include the proof and a general statement here for completeness:

Theorem 5.1. *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 $L^k(B) \le ab^k$, where $b \ge 2$. 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 $\mathcal{C}$,[6]

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

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

To prove Theorem 1.1 we set $\mathcal{C}$ to be the class of all 3OBPs of length at most n. Theorem 3.1 gives a bound corresponding to $a = O(n^2)$ and $b = O(\log n)$. This gives the required generator. The statements of Theorems 1.1 and 5.1 differ in that Theorem 5.1 bounds the error of the pseudorandom generator with respect to a matrix-valued function, while Theorem 1.1 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 following lemma gives the basis of our pseudorandom generator.

Lemma 5.2. *Let a, b, and $\mathcal{C}$ be as in Theorem 5.1 and $B \in \mathcal{C}$. Let $\varepsilon \in (0,1)$. Let T be a random variable over $\{0,1\}^n$ that is δ-almost $2k$-wise independent and each bit has expectation p, where we require*

$$p \le 1/(2b), \qquad k \ge \log_2\left(8an^4w/\varepsilon\right), \qquad \text{and} \qquad \delta \le 1/(2b)^{2k}.$$

[6] 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.

Let U be uniform over $\{0,1\}^n$. Let X be a μ-biased random variable over $\{0,1\}^n$ with $\mu \le \varepsilon/2ab^k$. Then

$$\left|\left|\mathop{\mathbb{E}}_{T,X,U}[B[\mathrm{Select}(T,X,U)]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 \le \varepsilon.$$

Proof. For a fixed $t \in \{0,1\}^n$, we have

$$\begin{aligned}
\left|\left|\mathop{\mathbb{E}}_{X,U}[B[\mathrm{Select}(t,X,U)]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 &= \left|\left|\mathop{\mathbb{E}}_{X}[B|_t[X]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 \\
&= \left|\left|\sum_{s\neq 0} \widehat{B|_t}[s]\widehat{X}(s)\right|\right|_2 \\
&\le \sum_{s\neq 0} \left|\left|\widehat{B|_t}[s]\right|\right|_2 |\widehat{X}(s)| \\
&\le L(B|_t)\mu,
\end{aligned}$$

Conditioning on whether or not $L^{\ge k}(B|_t) > 1$, we have

$$\begin{aligned}
\left|\left|\mathop{\mathbb{E}}_{T,X,U}[B[\mathrm{Select}(T,X,U)]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 \le{}& \mathop{\mathbb{P}}_{T}\left[L^{\ge k}(B|_T) > 1\right] \max_t \left|\left|\mathop{\mathbb{E}}_{X,U}[B[\mathrm{Select}(t,X,U)]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 \\
&+ \mathop{\mathbb{P}}_{T}\left[L^{\ge k}(B|_T) \le 1\right] \mu \mathop{\mathbb{E}}_{T}\left[L(B|_T) \mid L^{\ge k}(B|_T) \le 1\right].
\end{aligned}$$

We have

$$\mathrm{L}^{<k}(B) \le \sum_{1\le k'<k} ab^{k'} = ab\frac{b^{k-1}-1}{b-1} \le ab^k - 1.$$

Thus $\mathop{\mathbb{E}}_{T}\left[L(B|_T) \mid L^{\ge k}(B|_T) \le 1\right] \le ab^k$. Lemma 4.3 gives

$$\mathop{\mathbb{P}}_{T}\left[L^{\ge k}(B|_T) > 1\right] \le n^4 \cdot \frac{2a}{2^k}.$$

For all t, x, y, we have $\left|\left|B[\mathrm{Select}(t,x,y)] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 \le 2w$. Thus

$$\begin{aligned}
\left|\left|\mathop{\mathbb{E}}_{T,X,U}[B[\mathrm{Select}(T,X,U)]] - \mathop{\mathbb{E}}_{U}[B[U]]\right|\right|_2 &\le n^4 \cdot \frac{2a}{2^k} \cdot 2w + 1 \cdot \mu \cdot ab^k \\
&\le \frac{4an^4w}{8an^4w/\varepsilon} + ab^k\frac{\varepsilon}{2ab^k} \\
&\le \varepsilon.
\end{aligned}$$

□

Now we use the above results to construct our pseudorandom generator.

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 $\mu \leq \varepsilon'/2ab^k$. [7]
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 $\text{Select}(T, X, \tilde{U}) \in \{0,1\}^n$.[8]

The analysis of the algorithm proceeds roughly as follows.

- We have $p = \Theta(1/b)$, $\varepsilon' = \Theta(\varepsilon/wb\log n)$, $k = \Theta(\log(abwn/\varepsilon))$, $\delta = 1/b^{\Theta(k)}$, and $\mu = 1/b^{\Theta(k)}$.
- Every time we recurse, n is decreased to $\lfloor n(1-p/2) \rfloor$. After $O(\log(n)/p)$ recursions, n is reduced to $O(1)$. So the maximum recursion depth is $r = O(\log(n)/p) = O(b\log n)$.
- 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 and, hence, step 2.)
- The output is pseudorandom, as

$$\mathop{\mathbb{E}}_{U}\left[B[G_{a,b,n,\varepsilon}(U)]\right] = \mathop{\mathbb{E}}_{T,X,\tilde{U}}\left[B[\text{Select}(T,X,\tilde{U})]\right] \approx \mathop{\mathbb{E}}_{T,X,U}\left[B[\text{Select}(T,X,U)]\right] \approx \mathop{\mathbb{E}}_{U}\left[B[U]\right].$$

  The first approximate equality holds because we inductively assume that $\tilde{U}$ is pseudorandom. The second approximate equality holds by Lemma 5.2.

- The total seed length is the seed length needed to sample X and T at each level of recursion and $O(\log(1/\varepsilon')/p) = O(b\log(bwn/\varepsilon))$ truly random bits at the last level. Sampling X requires seed length $O(\log(n/\mu)) = O(k\log b)$ and sampling T requires seed length $O(k\log(1/p) + \log(1/\delta) + \log(1/p)\cdot\log(n\log(1/p))) = O(k\log b)$ so the total seed length is

$$r \cdot O(k\log b) + O(b\log(bwn/\varepsilon)) = O\left(b\cdot\log(b)\cdot\log(n)\cdot\log\left(\frac{abwn}{\varepsilon}\right)\right).$$

Lemma 5.3. *The probability that $G_{a,b,n,\varepsilon}$ fails at step 4 is bounded by $3\varepsilon'$—that is, $\mathop{\mathbb{P}}_{T}[|T| < pn/2] \leq 3\varepsilon'$.*

[7] 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.

[8] 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.

Proof. By a Chernoff bound for limited independence (Lemma A.1),

$$\mathop{\mathbb{P}}_{T}[|T| < pn/2] \le \left(\frac{20k'}{(1/2)^2 pn}\right)^{\lfloor k'/2 \rfloor} + 2\delta \cdot \left(\frac{n}{(1/2)pn}\right)^{k'},$$

where $k' \le 2k$ even is arbitrary. Set $k' = 2\lceil \log_2(1/\varepsilon') \rceil$. Step 2 ensures that $n > 160k'/p$ and our setting of δ gives that $\delta \le \varepsilon'(p/2)^{k'}$. Thus we have

$$\mathop{\mathbb{P}}_{T}[|T| < pn/2] \le 2^{-\log_2(1/\varepsilon')} + 2\varepsilon' \le 3\varepsilon'.$$

□

The following bounds the error of $G_{a,b,n,\varepsilon}$.

Lemma 5.4. *Let $B \in \mathcal{C}$. Then*

$$\left\| \mathop{\mathbb{E}}_{U_{s_{n,\varepsilon}}} \left[B[G_{n,\varepsilon}(U_{s_{n,\varepsilon}})]\right] - \mathop{\mathbb{E}}_{U}[B[U]] \right\|_2 \le 7wr\varepsilon',$$

where $r = O(\log(n)/p)$ is the maximum recursion depth of $G_{a,b,n,\varepsilon}$.

Proof. For $0 \le i < r$, let n_i, T_i, X_i, and $\tilde{U}_i$ be the values of n, T, X, and $\tilde{U}$ at recursion level i. We have $n_{i+1} = \lfloor n_i(1-p/2) \rfloor \le n(1-p/2)^{i+1}$ and $\tilde{U}_{i-1} = \mathrm{Select}(T_i, X_i, \tilde{U}_i)$. Let Δ_i be the error of the output from the i^{th} level of recursion—that is,

$$\Delta_i := \max_{B' \in \mathcal{C}} \left\| \mathop{\mathbb{E}}_{T_i, X_i, \tilde{U}_i} \left[B'[\mathrm{Select}(T_i, X_i, \tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[U]\right] \right\|_2.$$

Since the last level of recursion outputs uniform randomness, $\Delta_r = 0$. For $0 \le i < r$, we have, for some $B' \in \mathcal{C}$,

$$\begin{aligned}
\Delta_i &\le \left\| \mathop{\mathbb{E}}_{T_i, X_i, \tilde{U}_i} \left[B'[\mathrm{Select}(T_i, X_i, \tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[U]\right] \right\|_2 \cdot \mathop{\mathbb{P}}_{T}[|T| \ge pn/2] \\
&\quad + 2w \cdot \mathop{\mathbb{P}}_{T}[|T| < pn/2] \\
&\le \left\| \mathop{\mathbb{E}}_{T_i, X_i, \tilde{U}_i} \left[B'[\mathrm{Select}(T_i, X_i, \tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{T_i, X_i, U} \left[B'[\mathrm{Select}(T_i, X_i, U)]\right] \right\|_2 \\
&\quad + \left\| \mathop{\mathbb{E}}_{T_i, X_i, U} \left[B'[\mathrm{Select}(T_i, X_i, U)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[U]\right] \right\|_2 \\
&\quad + 2w \cdot \mathop{\mathbb{P}}_{T}[|T| < pn/2]
\end{aligned}$$

By Lemma 5.2,

$$\left\| \mathop{\mathbb{E}}_{T_i, X_i, U} \left[B'[\mathrm{Select}(T_i, X_i, U)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[U]\right] \right\|_2 \le \varepsilon'.$$

By Lemma 5.3,

$$\mathop{\mathbb{P}}_{T}[|T| < pn/2] \leq 3\varepsilon'.$$

We claim that

$$\left|\left|\mathop{\mathbb{E}}_{T_i,X_i,\tilde{U}_i}\left[B'[\text{Select}(T_i,X_i,\tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{T_i,X_i,U}\left[B'[\text{Select}(T_i,X_i,U)]\right]\right|\right|_2 \leq \Delta_{i+1}.$$

Before we prove the claim, we complete the proof: This gives $\Delta_i \leq \Delta_{i+1} + \varepsilon' + 2w \cdot 3\varepsilon'$. It follows that $\Delta_0 \leq 7wr\varepsilon'$, as required.

To prove the claim, consider *any* fixed $T_i = t$ and $X_i = x$. We have

$$\left|\left|\mathop{\mathbb{E}}_{\tilde{U}_i}\left[B'[\text{Select}(t,x,\tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[\text{Select}(t,x,U)]\right]\right|\right|_2 \leq \Delta_{i+1}.$$

Consider $\overline{B}_{x,t}[y] := B'[\text{Select}(t,x,y)]$ as a function of $y \in \{0,1\}^{n_i-|t|}$. Then $\overline{B}_{x,t}$ is a width-3 read-once oblivious branching program of length-$(n_i - |t|)$.

We inductively know that $\tilde{U}_i$ is pseudorandom for $\overline{B}_{x,t}$—that is, $\left|\left|\mathop{\mathbb{E}}_{\tilde{U}_i}\left[\overline{B}_{x,t}[\tilde{U}_i]\right] - \mathop{\mathbb{E}}_{U}\left[\overline{B}_{x,t}[U]\right]\right|\right|_2 \leq \Delta_{i+1}$. Thus

$$\left|\left|\mathop{\mathbb{E}}_{\tilde{U}_i}\left[B'[\text{Select}(t,x,\tilde{U}_i)]\right] - \mathop{\mathbb{E}}_{U}\left[B'[\text{Select}(t,x,U)]\right]\right|\right|_2 = \left|\left|\mathop{\mathbb{E}}_{\tilde{U}_i}\left[\overline{B}_{x,t}[\tilde{U}_i]\right] - \mathop{\mathbb{E}}_{U}\left[\overline{B}_{x,t}[U]\right]\right|\right|_2 \leq \Delta_{i+1},$$

as required. □

Proof of Theorem 5.1. Since $\varepsilon' \leq \varepsilon/(7wr)$, Lemma 5.4 implies that $G_{a,b,n,\varepsilon}$ has error at most ε. The seed length is

$$s_{a,b,n,\varepsilon} = O\left(b \cdot \log(b) \cdot \log(n) \cdot \log\left(\frac{abwn}{\varepsilon}\right)\right)$$

as required. □

6 Further Work

Our results hinge on the fact that "mixing" is well-understood for regular branching programs [9, 34, 27, 14, 40] and for (non-regular) width-2 branching programs [5]. 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 3.2 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 1.3 and other results.

The biggest obstacle to extending our techniques to $w > 3$ is Lemma 3.7. 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 3.2, 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. [9] and those of Brody and Verbin [10] and Steinberger [39]. We would like to combine them more cleanly – presently the proof is split into two parts (Proposition 3.2 and Lemma 3.7). 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 6.1 ([15, 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 [15, Proposition 2.7]. That is, any function f fooled by all small bias generators has sandwiching polynomials satisfying the hypotheses of Proposition 6.1.

Gopalan et al. [18] 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. 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] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *FOCS*, pages 276–287, Nov 1994.

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

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

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

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

[8] 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.

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

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

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

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

[13] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[31] Jelani Nelson. Exponential tail bounds without the moment generating function. `http://mathoverflow.net/questions/144396/exponential-tail-bounds-without-the-moment-generating-func`

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

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

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

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

[36] J. Schmidt, A. Siegel, and A. Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM J. Discrete Mathematics*, 8(2):223–250, 1995.

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

[38] D. Sivakumar. Algorithmic derandomization via complexity theory. In *CCC*, page 10, 2002.

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

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

[41] 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.

[42] 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.

A Chernoff Bound for Limited Independence

Lemma A.1 (Chernoff Bound for Limited Independence)**.** *Let $X_1 \cdots X_\ell$ be δ-almost k-wise independent random variables with $X_i \in \{0,1\}$ for all i. Set $X = \sum_i X_i$ and $\mu = \sum_i \mu_i = \sum_i \mathop{\mathbb{E}}_X [X_i]$, and suppose $\mu_i \leq 1/2$ for all i. If $k \leq \mu/10$ is even, then, for all $\alpha \in (0,1)$,*

$$\mathop{\mathbb{P}}_X [|X - \mu| \geq \alpha\mu] \leq \left(\frac{20k}{\alpha^2\mu}\right)^{\lfloor k/2 \rfloor} + 2\delta \cdot \left(\frac{\ell}{\alpha\mu}\right)^k.$$

The following proof is based on [36, Theorem 4]. The only difference is that we extend to almost k-wise independence from k-wise independence.

Proof. Assume, without loss of generality, that k is even. It is well-known [36, 31, 3] that, if the X_is are fully independent, then

$$\mathop{\mathbb{E}}_{X_i} \left[(X - \mu)^k\right] \leq (20k\mu)^{k/2}.$$

This also holds when the X_is are only k-wise independent, as $(X - \mu)^k$ is a degree-k polynomial in the X_is. Here the X_is are δ-almost k-wise independent, which gives

$$\mathop{\mathbb{E}}_{X_i} \left[(X - \mu)^k\right] \leq (20k\mu)^{k/2} + 2\delta\ell^k.$$

Thus we can apply Markov's inequality to obtain the result:

$$\mathop{\mathbb{P}}_X [|X - \mu| \geq \alpha\mu] = \mathop{\mathbb{P}}_X \left[(X - \mu)^k \geq (\alpha\mu)^k\right] \leq \frac{\mathop{\mathbb{E}}_{X_i} \left[(X - \mu)^k\right]}{(\alpha\mu)^k} \leq \left(\frac{20k\mu}{\alpha^2\mu^2}\right)^{k/2} + 2\delta \left(\frac{\ell}{\alpha\mu}\right)^k.$$

□

B First-Order Fourier Coefficients of Branching Programs

Theorem B.1. *Let B be a width-w, length-n, read-once, oblivious branching program. Then*

$$\sum_{i \in [n]} \left|\left|\widehat{B}[\{i\}]\right|\right|_2 \leq O(\log n)^{w-2}.$$

The proof is similar to the proof of the Coin Theorem by Steinberger [39]. The main difference is that we need a new proof of the collision lemma:

We call a layer B_i of a branching program **trivial** if $L(B_i) = 0$ and nontrivial otherwise. We say that a layer B_i has a **collision** if there exist two edges with the same label and the same endpoint, but different start points. All non-permutation layers have a collision. If there is a collision in layer i, then with probability at least $1/2$ a random restriction of layer i reduces the width.

Lemma B.2 (Collision Lemma)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a function computed by a width-w ordered branching program B. Then there exists a function g computed by a width-w ordered branching program B' such that every nontrivial layer of B' has a collision and*

$$\sum_i |\widehat{f}[\{i\}]| \leq \sum_i |\widehat{g}[\{i\}]|.$$

Proof. We will construct B' by flipping edge labels in B.

Begin by ordering the vertices in each layer by their acceptance probability – that is, the probability that $f(U) = 1$ conditioned on passing through that vertex. (If two vertices have the same acceptance probability order them arbitrarily.) We will flip the edge labels such that for every vertex the 0-edge leads to a higher-ranked vertex than the 1-edge (or they lead to the same vertex).

Note that flipping the edges in layer i only affects the corresponding Fourier coefficient. Thus we need only show that flipping the edges in layer i does not decrease $|\widehat{f}[i]|$.

Fix a layer i. For a vertex u on the left of layer i of B, let u_0 and u_1 be the vertices led to by the 0- and 1-edges respectively and let p_u be the probability that a random walk in B reaches u. For a vertex v on the right of layer i of B, let q_v be the acceptance probability of B under uniform input conditioned on passing through vertex v. Then

$$\widehat{f}[\{i\}] = \frac{1}{2} \sum_u p_u (q_{u_0} - q_{u_1}).$$

Flipping edge labels corresponds to flipping the signs of the terms in the above sum. Clearly, $|\widehat{f}[\{i\}]|$ is maximised if every term has the same sign. Our choice of flips ensures this is the case, as $q_{u_0} \geq q_{u_1}$.

Every nontrivial layer of B must have a collision, as a result of the ordering of edge labels: Consider a layer i and let v be the highest ranked vertex on its right such that the incoming edges are from different vertices on the left. Suppose, for the sake of contradiction, that the incoming edges have different labels. Pick the edge labelled 1 and let u be its start point. Let u_0 be the vertex reached from u by the edge labelled 0. Then, by our choice of labels u_0 is ranked higher than u – a contradiction, as u is the highest ranked vertex with distinct incoming edges. □

Define $\xi(n, w)$ to be the maximal first-order Fourier mass of any function computed by a length-n, width-w, ordered branching program. We follow the structure of Steinberger's proof [39] to bound ξ.

Lemma B.3. *For all n and $w \geq 3$, $\xi(n, w) \leq (2 + 2\log_2(n)) \cdot (\xi(n, w-1) + 1)$.*

Proof. Let B be a length-n, width-w, ordered branching program that maximises the first-order Fourier mass of the function f it computes. By Lemma B.2, we may assume that every nontrivial layer of B has a collision. We may assume that there are no trivial layers: otherwise we can remove them without affecting the Fourier mass.

Let $m = \lceil 1 + 2\log_2 n \rceil$. Split the first-order Fourier coefficients into m groups of the form

$$G_{i'} = \{i \in [n] : i \bmod m = i'\}.$$

We bound the first-order Fourier mass of each group separately and sum them together. i.e. $\sum_{i\in[n]} |\widehat{f}[\{i\}]| = \sum_{i'\in[m]} \sum_{i\in G_{i'}} |\widehat{f}[\{i\}]|$. Fix one group $G = G_{i'}$.

We apply a random restriction to B to obtain the function $f|_{\overline{G}\leftarrow U}$ computed by the branching program $B|_{\overline{G}\leftarrow U}$. We have

$$\sum_{i\in G} |\widehat{f}[\{i\}] \le \mathop{\mathbb{E}}_{U} \left[\sum_{i\in G} |\widehat{f|_{\overline{G}\leftarrow U}}[\{i\}]| \right].$$

So if suffices to bound the first-order Fourier mass of $f|_{\overline{G}\leftarrow U}$.

We claim that $B|_{\overline{G}\leftarrow U}$ is a width-$(w-1)$, ordered branching program with probability at least $1 - n \cdot 2^{1-m}$. This implies that

$$\mathop{\mathbb{E}}_{U} \left[\sum_{i\in G} |\widehat{f|_{\overline{G}\leftarrow U}}[\{i\}]| \right] \le \xi(n, w-1) + n \cdot 2^{1-w} \cdot n \le \xi(n, w-1) + 1.$$

Thus

$$\sum_{i\in[n]} |\widehat{f}[\{i\}]| \le \sum_{i'\in[m]} \mathop{\mathbb{E}}_{U} \left[\sum_{i\in G_{i'}} |\widehat{f|_{\overline{G_{i'}}\leftarrow U}}[\{i\}]| \right] \le \sum_{i'\in[m]} \xi(n, w-1) + 1 \le m(\xi(n, w-1) + 1),$$

as required.

Now to prove the claim: Fix an unrestricted layer i of $B|_{\overline{G}\leftarrow U}$ other than the last layer (which can always be assumed to have width 2 anyway). Layer i is followed by $m-1$ restricted layers. With probability at least $1 - 2^{1-m}$ at least one of these layers will contain a collision, thus reducing the number of vertices on the right of layer i. A union bound gives the required probability. □

Lemma B.4. *$\xi(n, 2) \le 10$.*

Solving the recurrance for ξ gives $\xi(n, w) \le O(\log n)^{w-2}$, as required.

C Optimality of Result

The following result shows that Theorem 3.1 is close to optimal.

Proposition C.1. *There exists an infinite family of functions $f_n : \{0,1\}^n \to \{0,1\}$ and that are computed by 3OBPs such that the following holds. Let $q : \mathbb{N} \to \mathbb{R}$ be an increasing function with $20 \le q(n) \le \exp(\exp(o(\sqrt{\log n})))$. For all sufficiently large n, there exists $k \in [n]$ such that*

$$L^k(f_n) > q(n) \cdot \left(\frac{\log n}{20 \log\log q(n)} \right)^k.$$

.

Our main result shows that $L^k(f_n) \le \mathrm{poly}(n) \cdot (O(\log n))^k$. Setting $q(n) = \mathrm{poly}(n)$, this proposition shows that the base $O(\log n)$ cannot be improved by more than a $\log\log n$ factor. The $\log\log n$ factor comes from the fact that we allow a polynomial factor $q(n)$ in the bound. If we demand $q(n) = O(1)$, the base $O(\log n)$ is optimal.

Proof. Let $n = m \cdot 2^m$, where m is an integer. Define $f_n : \{0,1\}^n \to \{0,1\}$ by

$$f_n(x) = \prod_{i \in [2^m]} \left(1 - \prod_{j \in [m]} x_{i,j}\right),$$

where we view $x \in \{0,1\}^n$ as a $2^m \times m$ matrix $x \in \{0,1\}^{2^m \times m}$. This is (up to a negation) the Tribes function [4]. This function can be computed by a 3OBP. Now we show that it has large Fourier growth.

The Fourier coefficients of f_n are given as follows. For $s \subset [n]$ (which we identify with $s \in \{0,1\}^{2^m \times m}$),

$$\begin{aligned}
\widehat{f_n}[s] =& \mathop{\mathbb{E}}_U [f(U)\chi_s(U)] \\
=& \mathop{\mathbb{E}}_U \left[\prod_{i \in [2^m]} \left(1 - \prod_{j \in [m]} U_{i,j}\right) \chi_{s_i}(U_i)\right] \\
=& \prod_{i \in [2^m]} \mathop{\mathbb{E}}_U \left[\chi_{s_i}(U_i) - \prod_{j \in [m]} U_{i,j}\chi_{s_i}(U_i)\right] \\
=& \prod_{i \in [2^m]} \left(\mathbb{I}(s_i = 0) - 2^{-m}(-1)^{|s_i|}\right),
\end{aligned}$$

where $s_i = (s_{i,1}, s_{i,2}, \cdots, s_{i,m})$. The damped Fourier mass is also easy to compute. For $p \in (0,1)$,

$$\begin{aligned}
L_p(f_n) + |\widehat{f_n}[0]| =& \sum_{s \in \{0,1\}^{2^m \times m}} p^{|s|} |\widehat{f_n}[s]| \\
=& \sum_{s \in \{0,1\}^{2^m \times m}} \prod_{i \in [2^m]} p^{|s_i|} \left|\mathbb{I}(s_i = 0) - 2^{-m}(-1)^{|s_i|}\right| \\
=& \prod_{i \in [2^m]} \sum_{s_i \in \{0,1\}^m} p^{|s_i|} \left|\mathbb{I}(s_i = 0) - 2^{-m}(-1)^{|s_i|}\right| \\
=& \prod_{i \in [2^m]} \left(1 - 2^{-m} + \sum_{s_i \neq 0} p^{|s_i|} 2^{-m}\right) \\
=& \prod_{i \in [2^m]} \left(1 - 2^{-m} + 2^{-m}(1+p)^m - 2^{-m}\right) \\
=& \left(1 + \frac{(1+p)^m - 2}{2^m}\right)^{2^m}.
\end{aligned}$$

Set $p = (1 + \log(3 + \log q(n)))/m$. We have

$$(1+p)^m = \left(1 + \frac{1 + \log(3 + \log q(n))}{m}\right)^m = e^{1+\log(3+\log q(n))}(1 - o(1)) \geq 3 + \log q(n)$$

for sufficiently large m. Thus, for sufficiently large m,

$$L_p(f_n) \geq \left(1 + \frac{(1+p)^m - 2}{2^m}\right)^{2^m} - 1 \geq \left(1 + \frac{1 + \log q(n)}{2^m}\right)^{2^m} - 1 = e^{1+\log q(n)}(1 - o(1)) - 1 \geq q(n).$$

Suppose for the sake of contradiction that $L^k(f_n) \le q(n) \cdot (\log n/20 \log\log q(n))^k$ for all $k \in [n]$. We have

$$L_p(f_n) = \sum_{k\in[n]} p^k L^k(f_n) \le \sum_{k\in[n]} \left(\frac{1+\log(3+\log q(n))}{m}\right)^k \cdot q(n) \cdot \left(\frac{\log n}{20\log\log q(n)}\right)^k \le \sum_{k\in[n]} q(n) 2^{-k} < q(n),$$

which is a contradiction. □

A more careful analysis gives the following bound.

Proposition C.2. *There exists an infinte family of functions $f_n : \{0,1\}^n \to \{0,1\}$ that are computed by 3OBPs such that, for all $k \in [n]$,*

$$L^k(f) \ge \Omega\left(\frac{\log n}{\log k}\right)^k.$$

Proof. Let n, m, and f_n be as in the proof of Proposition C.1. For $s \in \{0,1\}^{2^m \times m}$, denote

$$\ell(s) = |\{i \in [2^m] : s_i \neq 0\}| = |\{i \in [2^m] : \exists j \in [m] \;\; s_{i,j} = 1\}|.$$

Then, for all $s \in \{0,1\}^{n\times m}$, we have

$$|\widehat{f_n}[s]| = \prod_{i\in[2^m]} \left|\mathbb{I}(s_i = 0) - 2^{-m}(-1)^{|s_i|}\right| = (1-2^{-m})^{2^m-\ell(s)} \cdot (2^{-m})^{\ell(s)}.$$

Fix ℓ with $k/\ell \le m$. Set $h = \lfloor k/\ell \rfloor$. Choose $i, j \ge 0$ with $i + j = \ell$ and $ih + j(h+1) = k$. Then

$$\begin{aligned}
L^k(f_n) \ge& \sum_{|s|=k \wedge \ell(s)=\ell} |\widehat{f_n}[s]| \\
\ge& \binom{2^m}{\ell}\binom{m}{h}^i \binom{m}{h+1}^j \cdot (1-2^{-m})^{2^m-\ell} \cdot (2^{-m})^\ell \\
\ge& \left(\frac{2^m}{\ell}\right)^\ell \left(\frac{m}{h}\right)^{hi} \left(\frac{m}{h+1}\right)^{(h+1)j} \cdot \left(1-\frac{1}{2^m}\right)^{2^m} \cdot \left(\frac{1}{2^m}\right)^\ell \\
\ge& \frac{1}{4}\left(\frac{1}{\ell}\right)^\ell \left(\frac{m}{h}\right)^{hi} \left(\frac{m}{h+1}\right)^{(h+1)j} \\
\ge& \frac{1}{4}\left(\frac{1}{\ell}\right)^\ell \left(\frac{m}{h+1}\right)^{hi+(h+1)j} \\
\ge& \frac{1}{4}\left(\frac{1}{\ell}\right)^\ell \cdot \left(\frac{m}{k/\ell+1}\right)^k.
\end{aligned}$$

Suppose $k \le 2^{m-1}$. Setting $\ell = \lceil k/\log_2(2k)\rceil$, we have

$$\begin{aligned}
L^k(f_n) \ge& \frac{1}{4}\left(\frac{1}{\ell}\right)^\ell \left(\frac{m}{k/\ell+1}\right)^k \\
\ge& \frac{1}{4} \cdot \frac{1}{2^{2k+2}} \cdot \left(\frac{m}{\log_2 k + 2}\right)^k,
\end{aligned}$$

as

$$\log_2(\ell^\ell) = \ell \log_2 \ell < \frac{k + \log_2 k}{\log_2 k} \log_2(k + \log_2 k) \leq 2k + 2.$$

Since $m = \Theta(\log n)$, this gives the result for $k \leq 2^{m-1}$. If $k > 2^{m-1}$, then $\log k = \Theta(\log n)$ and the result is trivial. □

ECCC ISSN 1433-8092

http://eccc.hpi-web.de