Electronic Colloquium on Computational Complexity, Report No. 83 (2012)

# Pseudorandomness for Permutation Branching Programs Without the Group Theory

Thomas Steinke[*]
Harvard School of Engineering and Applied Sciences
tsteinke@seas.harvard.edu

Friday 29th June, 2012

**Abstract**

We exhibit an explicit pseudorandom generator that stretches an $O\left(\left(w^4 \log w + \log(1/\varepsilon)\right) \cdot \log n\right)$-bit random seed to $n$ pseudorandom bits that cannot be distinguished from truly random bits by a permutation branching program of width $w$ with probability more than $\varepsilon$. This improves on the seed length $O\left(\left((w!)^{11} + \log(1/\varepsilon)\right) \cdot \log n\right)$ of Koucký, Nimbhorkar, and Pudlák [8] and $O\left(\left(w^8 + \log(1/\varepsilon)\right) \cdot \log n\right)$ of De [4]. More importantly, the analysis of our generator uses only combinatorial and linear-algebraic arguments, whereas the previous proofs refer to groups or representation theory.

## 1 Introduction

This paper is about derandomizing small space computations—that is, we wish to simulate randomized computations using deterministic ones without significantly increasing the space required. Thus we hope to understand the relationship between two important computational resources: randomness and space.

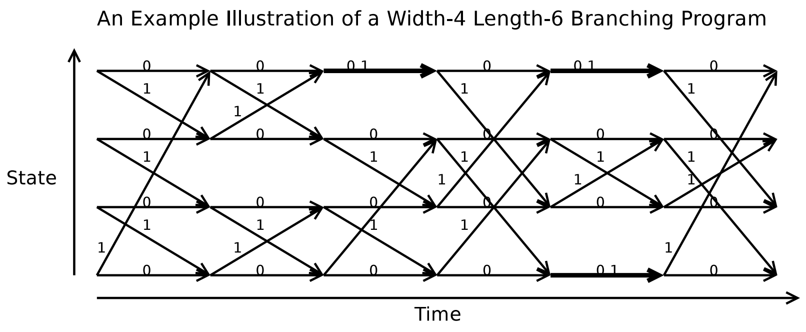
The typical way we derandomize a randomized computation is to replace the truly random bits used by the computation with suitable pseudorandom bits. We require that using pseudorandom, rather than truly random, bits does not change the output of the computation with probability more than $\varepsilon$; we call this parameter the **error**. The algorithm that produces the pseudorandom bits is called a **pseudorandom generator** and the number of random bits needed by this algorithm to sample the distribution is called the **seed length**. The pseudorandom generator needs to be efficiently computable and the seed length needs to be short enough for all possible seeds to be enumerated in a deterministic simulation.

The model of (nonuniform) small-space computation we use is (oblivious read-once) **branching programs**. A length-$n$ width-$w$ branching program can be used to represent a computation that has $\log w$ bits of space and uses $n$ random bits. (The actual input to the computation is captured

[*]This work was supported by the Lord Rutherford Memorial Research Fellowship and NSF grant CCF-1116616.

ISSN 1433-8092

in the nonuniformity of the model.) The branching program has $n$ time steps and $w$ states; the $i^{\text{th}}$ step is given by a function $B_i : \{0,1\} \times [w] \to [w]$ that takes the $i^{\text{th}}$ random input bit $x$ and the state $u$ at time $i-1$ and maps it to a new state $B[x](u)$ at time $i$. It is helpful to view each step of the branching program as a bipartite graph with $w$ vertices on each side and edges of the form $(u, B[x](u))$ labelled by the bit $x$. The whole branching program can then be viewed as a directed graph with $n$ layers.


An Example Illustration of a Width-4 Length-6 Branching Program
State
Time
0
1
0 1

A major goal for derandomization research is to prove that any randomized computation can be simulated by a deterministic computation with only a constant factor increase in space usage—in particular, $\mathcal{RL} = \mathcal{L}$. This problem has a long history [7, 9, 15, 13, 1]. To prove this, it suffices to construct a pseudorandom generator for length-$n$ width-$2^{O(\log n)}$ branching programs that has seed length $O(\log n)$. However, so far the best seed length is $O(\log^2 n)$ [9, 7] (although the best derandomization result is $\mathcal{RL} \subset \mathcal{L}^{3/2}$ [15], which uses Nisan's pseudorandom generator [9] in a very sophisticated way). These general results have been unbeaten for over a decade.

Stronger results are known for restricted classes of branching programs. In particular, we consider sublogarithmic space (that is, subpolynomial width)—usually constant space and width. Moreover, we focus on regular and permutation branching programs: A **regular branching program** satisfies the condition that each output of the $i^{\text{th}}$ step $B_i : \{0,1\} \times [w] \to [w]$ corresponds to two inputs—that is, for each $i \in [n]$ and $v \in [w]$, there are exactly two pairs $(x,u) \in \{0,1\} \times [w]$ with $B_i[x](u) = v$. Equivalently, each step in the branching program corresponds to a 2-regular bipartite graph. A **permutation branching program** has the additional constraint that these pairs are of the form $(0, u_0)$ and $(1, u_1)$ or, equivalently, $B_i[x]$ is a permutation on $[w]$ for each fixed $i$ and $x$. In the graph of a permutation branching program, the edges labelled 0 form a matching in each step and the edges labelled 1 form another matching.

Intuitively, a regular branching program is one that monotonically "absorbs" randomness as the computation progresses. More formally, the entropy of the program state monotonically increases (or stays the same) every time a random input bit is read. A permutation branching program has an even stronger property: the randomness of the state will monotonically increase even when given imperfect random bits (subject to certain constraints). More interestingly, a permutation branching program can be "run backwards." These properties are useful because there is an interesting relationship between the randomness a program absorbs and the error it accumulates when given pseudorandom input—this relationship is a central theme in this paper.

A major objective is to construct a pseudorandom generator for constant-width length-$n$ regular branching programs with seed length $O(\log n)$ for constant error; presently $O(\log n \cdot \log\log n)$ is the best known seed length [2, 3]. We give a pseudorandom generator with seed length $O(\log n)$ for constant-width *permutation* branching programs and we look at generalising this result to the regular case.

## 1.1 INW Generator

Our results are based on the Impagliazzo-Nisan-Wigderson (INW) generator [7]. Many results (including [14, 2, 3, 8, 4, 10] to name a few) have been based on this generator.

Intuitively, the INW generator 'recycles' randomness: The basic observation underlying its analysis is as follows

> If, at time $t$, the program has used $t$ random bits and only has $s$ bits of space, then it must have 'forgotten' $t - s$ of those bits. Hence, those forgotten bits can be 'recycled' and used by the program in later time steps.

The randomness extraction properties of expander graphs are used to do this recycling. This basic observation essentially allows us to halve the amount of random bits used by the program. Recursively repeating this construction $\log n$ many times produces the INW generator.

The key parameter of the INW generator is the second eigenvalue $\lambda_H \in (0, 1)$ of the expander graphs used. (See [16, 6, 11] for more background on expander graphs.) Each application of the expander requires $O(\log(1/\lambda_H))$ random bits, giving total seed length $O(\log(1/\lambda_H) \cdot \log n)$ for the INW generator. And the recycled bits are $O(w \cdot \lambda_H)$ close to uniform in statistical distance, where $s = \log w$ bits have been 'remembered' and need to be replaced.

The difficult part of the analysis of the INW generator is bounding the total **error**—that is, we need to show that, for any branching program $B$ of the relevant type and size, replacing the truly random input of $B$ with the pseudorandom input from the INW generator does not change the output distribution of $B$ by more than $\varepsilon$ in statistical distance.

The original analysis of the INW generator shows that, for a width-$w$ length-$n$ branching program, the error is bounded by $O(n \cdot w \cdot \lambda_H)$: the $\lambda_H$ factor comes from the expander not perfectly recycling randomness; the $w$ factor comes from the $\log w$ bits that are 'remembered' by the program and need to be replaced; and the $n$ factor comes from the fact that the 'recycling' is performed $n$ times in total. (Subsequent results, discussed below, have shown that this analysis is pessimistic in some cases and the error does not grow linearly in $n$.)

The original analysis of the INW generator gives seed length $O((\log w + \log(1/\varepsilon) + \log n) \cdot \log n)$. Even for constant $w$ and $\varepsilon$, the seed length is $O(\log^2 n)$, rather than the desired $O(\log n)$. The original INW analysis matches the results for Nisan's generator [9]; however, the INW generator has shown to be more amenable to generalisation and improvement, as shown below.

## 1.2 Previous Work

The 20-year-old original analysis of the INW generator remains unbeaten, even for branching programs of width $w = 3$. (Some progress has recently been made on *hitting sets* for width-3 branching programs [17, 5].) However, restricting the analysis to regular or permutation branching programs allows significant improvement.

The results we discuss in this section all improve the error analysis of the INW generator for the special cases of regular and permutation branching programs. Specifically, they improve the dependence on $n$: they show that the error grows logarithmically for regular branching programs or is even independent of $n$ for permutation branching programs, rather than growing linearly, as in the original analysis.

Braverman, Rao, Raz, and Yehudayoff [2] show that most input bits for regular branching programs have small 'weight'—that is, they have little influence on the output of the program. Specifically, the total weight of all the bits is bounded by a polynomial in the width of the program (independent of the length $n$). Thus most bits have low weight and cannot contribute much error to the output of the program. This analysis suffices to prove that the INW generator works for width-$w$ length-$n$ regular branching programs with seed length $O\left((\log w + \log(1/\varepsilon) + \log\log n) \cdot \log n\right)$ for error $\varepsilon$. Our proof incorporates similar techniques, which we discuss in Section 3.1. Brody and Verbin [3] use a lower bound on the ability of branching programs to distinguish product distributions in order to achieve a similar seed length.

Building on [12, 11], Rozenman and Vadhan [14] analyse the behaviour of the INW generator on consistently labelled regular digraphs, which are simply permutation branching programs with each step being identical. They use this to give a different proof of Reingold's result [12] that undirected s-t connectivity can be solved in deterministic log space—that is, $\mathcal{SL} = \mathcal{L}$. (The difference between $\mathcal{SL}$ and $\mathcal{RL}$ is analogous to the difference between permutation branching programs and (regular) branching programs [13].) Their arguments, when applied to permutation branching programs, say that, if the program state steadily accumulates randomness (or, more formally, converges to its stationary distribution), then this convergence overpowers the error introduced by the INW generator, as long as the program is sufficiently long. We also use mixing to bound error accumulation in Section 3.2.

Koucký, Nimbhorkar, and Pudlák [8] show that the INW generator works for length-$n$ group product programs over a group $G$ with seed length $O\left(\left(|G|^{11} + \log(1/\varepsilon)\right) \cdot \log n\right)$ for error $\varepsilon$. Group products are a subclass of permutation branching programs with an algebraic structure; the size of the group corresponds to the width. By a reduction to the symmetric group, their work gives a pseudorandom generator for general permutation branching programs. However, this gives a seed length with an exponential dependence on the width, namely $O\left(\left((w!)^{11} + \log(1/\varepsilon)\right) \cdot \log n\right)$.

De [4] obtained an improved seed length of $O\left(\left(w^8 + \log(1/\varepsilon)\right) \cdot \log n\right)$ for error $\varepsilon$ and length-$n$ width-$w$ permutation branching programs. He also obtains seed length $O\left(\left(w^8 + \log(1/\varepsilon) + \log\log n\right) \cdot \log n\right)$ for regular branching programs.

## 1.3 Our Results

We give an improved analysis of the INW generator for permutation branching programs. Our main result is the following.

**Theorem 1** (Main Result). *The INW generator stretches a random seed of length $O\left(\left(w^4 \log w + \log(1/\varepsilon)\right) \cdot \log n\right)$ to $n$ pseudorandom bits that cannot be distinguished from truly* *random bits by a permutation branching program of width $w$ with probability greater than $\varepsilon$.*

More importantly, our analysis uses purely linear-algebraic and combinatorial arguments and completely avoids any reference to groups or representation theory, which are used in the proofs of Koucký et al. [8] and De [4]. Our approach yields an improved seed length, clarifies intuition and obstacles, and gives a shorter proof. Our analysis is inspired by that of Koucký, Nimbhorkar, and Pudlák [8], but was developed largely independently of De's, the full proof of which was not made available until June 2012.

Our analysis hinges on the two following ideas. These ideas are implicit in Koucký et al., but we make them explicit. Both ideas depend on the connection between randomness and error.

- **"Introducing Error Requires Introducing Randomness":** Every bit of the input has a certain amount of influence on the output of the program. This influence does two things: (i) it adds randomness to the program state and (ii) it can introduce error if the bit is not truly random. Crucially, a bit cannot do (ii) without doing (i). So error can only be introduced in conjunction with randomness. Since only log(width) bits of randomness can be introduced, we can bound the amount of error that is introduced this way. This is very similar to the argument used by Braverman et al. [2], although we are in a different context (permutation branching programs, rather than regular branching programs) and have slightly different intuition.
- **"Mixing Kills Error":** Introducing randomness (mixing) also reduces error. Randomizing or mixing the states makes the program 'forget' the past, thereby reducing the error. Whenever we introduce error, we also introduce randomness and this randomness reduces the error. The introduction of error and mixing balance out and error does not accumulate without bound. Cosmetically at least, this argument is similar to the known proofs that undirected s-t connectivity is in logspace ($\mathcal{SL} = \mathcal{L}$) [12, 14].

Much of the overall structure of our proof comes from Koucký et al.. However, we argue about permutation programs directly, rather than about group product programs. Crucially, our arguments are combinatorial and linear-algebraic, rather than group-theoretic: Rather than reasoning about cosets of subgroups, we reason about connected components. Rather than reasoning about the two-norm of a probability distribution over the group elements, we reason about the Frobenius norm of the stochastic matrix corresponding to a section of the program. Rather than reasoning about convolutions of distributions over group elements, we reason about composition of programs and matrix multiplication.

An important open problem is developing an explicit pseudorandom generator for constant-width length-$n$ regular branching programs with seed length $O(\log n)$. One hope for solving this problem is to generalise the results of Koucký et al. from permutation branching programs to regular branching programs. However, we cannot easily discuss non-permutation branching programs in the language of group theory. Thus the first step in generalising these results is to translate them into a language that is amenable to the analysis of regular branching programs. We do this; and this is our main contribution. We also show that many of the intermediate results in the proof hold for non-permutation regular branching programs. Thus we can pinpoint the obstacles to generalising the proof. (De also uses a linear-algebraic approach to reason about regular branching programs.)

Moreover, our linear-algebraic and combinatorial approach also clarifies the intuition behind the proof. In particular, it makes the connection to related works [2, 3, 12, 14] clearer.

## 2 Notations, Definitions, and Basic Facts

This section provides formal definitions as well as useful lemmas.

### 2.1 Branching Programs

Before we define branching programs, we consider different ways of viewing an arbitrary computation. We then define branching programs as a restricted class of programs.

- Intuitively, a program takes a start state and an input string and produces a final state. So we can view a program as a function as follows.

  Formally, a length-$n$ width-$w$ **program** is a function $B : \{0,1\}^n \times [w] \to [w]$. We denote $B$ evaluated at $x \in \{0,1\}^n$ and $u \in [w]$ by $B[x](u)$. We view the program $B$ as taking an input string $x$ and an initial state $u$ to a final state $B[x](u)$.

- In our applications, the input $x$ is a (pseudo)random string. So we also view programs as Markov chains represented by matrices, as is done in [4]. For each $x \in \{0,1\}^n$, we let $B[x] \in \{0,1\}^{w \times w}$ be a matrix defined by

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

  Then the matrix $B[x]$ represents a deterministic Markov chain that shows how $B$ maps initial states to final states with a given input $x$. Note that we use brackets rather than subscripts to index matrices or vectors; subscripts are reserved for denoting sequences.

  For a random variable $X$ on $\{0,1\}^n$, we define

  $$B[X] := \mathbb{E}\left[B[X]\right] \in \mathbb{R}^{w \times w}.$$

  Then $B[X](u,v)$ is the probability that $B$ takes the initial state $u$ to the final state $v$ when given a random input from the distribution $X$. We further overload notation by defining $B := B[U]$ to be a matrix where $U$ is uniform on $\{0,1\}^n$. Here the matrix $B$ represents a the action of the program $B$ with truly random input as the transition matrix of a Markov chain.

- A program can also be depicted as a labelled bipartite graph. Each initial state $u$ is a left vertex and each final state $v$ is a right vertex. There is an edge labelled $x$ from $u$ to $v$ if and only if $B[x](u) = v$. Running the program $B$ on a random input corresponds to taking a random outgoing edge from the start state vertex to the final state vertex.

However, we are not interested in arbitrary computations. We demand that the program reads one input bit at a time, rather than all at once. A branching program captures this restriction by demanding that the program be composed of several smaller programs. They are formally defined as follows.

Let $B$ and $B'$ be width-$w$ programs of length $n$ and $n'$ respectively. We define the **composition** $B \circ B' : \{0,1\}^{n+n'} \times [w] \to [w]$ 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. This corresponds to running $B$ then $B'$ on separate inputs, but $B$ passes information to $B'$ as the final state of $B$ becomes the initial state of $B'$. In the matrix view of programs, composition corresponds to matrix multiplication—that is, $B \circ B' = B \cdot B'$, where the two programs are composed on the left hand side and the two matrices are multiplied on the right hand side. We can also view this composition as a three-layer graph, where we take the bipartite graphs of $B$ and $B'$ and use the right vertices of $B$ as the left vertices of $B'$.

A length-$n$ width-$w$ **branching program** is a program $B$ that can be written $B = B_1 \circ B_2 \circ \cdots \circ B_n$, where each $B_i$ is a length-1 width-$w$ program.

Note that a branching program can be viewed as a directed acyclic graph: the vertices are in $n+1$ layers each containing $w$ vertices. Let $(u,i)$ be a vertex in layer $i$. Then the neighbours of $(u,i)$ are $(B_{i+1}[0](u), i+1)$ and $(B_{i+1}[1](u), i+1)$.

For a program $B$ and an arbitrary distribution $X$, the matrix $B[X]$ is **stochastic**—that is, $\sum_v B[X](u,v) = 1$ for all $u$ and $B[X](u,v) \geq 0$ for all $u$ and $v$. A program $B$ is called a **regular program** if the matrix $B$ is a **doubly stochastic** matrix—that is, $B$ and its transpose $B^*$ are stochastic. A program $B$ is called a **permutation program** if $B[x]$ is a permutation 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 composition.

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

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

## 2.2 Pseudorandom Generators

We are interested in analysing pseudorandom generators for (permutation) branching programs. A **pseudorandom generator** for a class $\mathcal{C}$ of programs is an *efficiently computable* family of functions $G_n : \{0,1\}^{s(n)} \to \{0,1\}^n$ such that, for *any* $B \in \mathcal{C}$, the output distribution of $B$ when given pseudorandom input is $\varepsilon$-close to the output distribution of $B$ when given truly random input—that is,

$$\left|\left|B[G_n(U_{s(n)})] - B\right|\right| \leq \varepsilon,$$

where $n$ is the length of $B$ and $U_{s(n)}$ is uniform on $\{0,1\}^{s(n)}$. Here $s(n)$ is the **seed length** of $G$ and $\varepsilon$ is the **error** of $G$. We say that $G$ **$\varepsilon$-fools** $\mathcal{C}$. In our applications $\mathcal{C}$ will be the class of length-$n$ width-$w$ permutation branching programs.

## 2.3 INW Generator

The pseudorandom generator we use is the Impagliazzo-Nisan-Wigderson (INW) generator, which is defined formally as follows.

Let $H$ be an explicit family of $2^d$-regular expander graphs with normalised second eigenvalue bounded by $\lambda_H \in (0,1)$. Let $H_n$ be the graph in the family $H$ of size $2^n$ and let $H_n(x,y)$ be the $y^{\text{th}}$ neighbour ($y \in \{0,1\}^d$) of $x \in \{0,1\}^n$ in the graph $H_n$. The optimal dependence of $d$ on $\lambda_H$ is $d = O(\log(1/\lambda_H))$ [11].

The **INW generator** of length $2^i$ is a function $G_i : \{0,1\}^{i \cdot d+1} \to \{0,1\}^{2^i}$ and is defined recursively by

$$G_0(x) = x \quad \text{and} \quad G_{i+1}(x,y) = (G_i(x), G_i(H_{i\cdot d+1}(x,y))).$$

To produce $n$ pseudorandom bits we use the output of $G_{\lceil \log n \rceil}$ (with uniform random bits as input), which has seed length $d \cdot \lceil \log n \rceil + 1 = O(\log(1/\lambda_H) \cdot \log n)$. Also note that $G_i$ can be computed in space $O(d \cdot \log n)$ [11].

## 2.4 Expander Product

The INW generator can also be viewed as repeated "pseudorandom composition" of programs.

We can express this pseudorandom composition as an operation on programs: Let $H$ be a family of $2^d$-regular graphs where $H_n$ has vertex set $\{0,1\}^n$. For $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, denote by $H_n(x,y) \in \{0,1\}^n$ the $y^{\text{th}}$ neighbour of $x$. We define the **expander product** $B \circ_H B' : \{0,1\}^{n+d} \times [w] \to [w]$ of length-$n$ width-$w$ programs $B$ and $B'$ with respect to $H$ by

$$(B \circ_H B')[x \circ y](u) := B'[H_n(x,y)](B[x](u)).$$

Define $J$ to be the complete graph on $\{0,1\}^n$ with $J(x,y) = y$, then $B \circ_J B' = B \circ B'$. An expander graph $H$ approximates $J$ and thus the expander product approximates composition—that is, the expander product is a "pseudorandom composition".

We can view the INW generator as an operation that maps a length-$n$ width-$w$ branching program $B$ to a length-$(d \cdot \log n + 1)$ width-$w$ program $\tilde{B}$, where $\tilde{B}[x] := B[G_{\log n}(x)]$. We can equivalently define the INW generator in terms of the expander product: If $B$ has length 1, then $\tilde{B}[x] := B[x]$. Otherwise, if $B = B_0 \circ B_1$, then $\tilde{B} = \tilde{B}_0 \circ_H \tilde{B}_1$, where $H$ is the family of $2^d$-regular expander graphs used by $G$.

It is also important that the expander product preserves the permutation property of programs like composition does.

**Lemma 2.** *Let $B$ and $B'$ be length-$n$ width-$w$ permutation programs. Let $H$ be a family of $2^d$-regular graphs. Then $B \circ_H B'$ is a permutation program.*

*Proof.* Fix $(x,y) \in \{0,1\}^n \times \{0,1\}^d$. By assumption, $B[x]$ and $B'[H_n(x,y)]$ are permutations. Thus $(B \circ_H B')[x \circ y] = B[x] \cdot B'[H_n(x,y)]$ is also a permutation. □

## 2.5 Spectral Analysis of Branching Programs

We reason about the connected components of branching programs. We will decompose the matrices corresponding to branching programs into a subspace that is uniform on each connected component of the branching program and a subspace that is orthogonal to uniform on these components.

Formally, for a doubly stochastic matrix matrix $B \in \mathbb{R}^{w \times w}$, define subspaces of $\mathbb{R}^w$ as follows

$$||B := \{x : ||xB||_2 = ||x||_2\}, \quad B|| := \{x : ||Bx||_2 = ||x||_2\}.$$

Also define $\perp B$ as the orthogonal complement of $||B$ and $B\perp$ as the orthogonal complement of $B||$.

The following lemma clarifies the meaning of these subspaces.

**Lemma 3.** *Let $B$ be a doubly stochastic $w \times w$ matrix. Define a graph $G(B)$ on $[w]$ where there is an edge between $u$ and $v$ if and only if $u$ and $v$ have a common neighbour in $B$—that is,*

$$(u, v) \in G(B) \iff \exists s \ (B(u, s) > 0 \wedge B(v, s) > 0).$$

*Equivalently, $(u, v) \in G(B)$ if and only if $(BB^*)(u, v) > 0$, where $B^*$ is the transpose of $B$. Note that this graph is undirected. Then $x \in ||B$ if and only if $x$ is constant on each connected component of $G(B)$—that is,*

$$x \in ||B \iff \forall (u, v) \in G(B) \ (x(u) = x(v)).$$

*And $y \in \perp B$ if and only if $y$ averages to zero on each connected component of $G(B)$. The same holds for $B||$ and $B\perp$ except with $G(B)$ replaced with $G^*(B) := G(B^*)$.*

We refer to the connected components of $G(B)$ from Lemma 3 as the **left connected components** of $B$. The **right connected components** of $B$ are the connected components of $G^*(B)$. We say that a doubly stochastic matrix $B$ is **connected** if it has only one left (or, equivalently, right) connected component.

*Proof.* We have

$$||xB||_2^2 = \sum_u \left( \sum_v x(v)B(v, u) \right)^2 \leq \sum_u \sum_v x(v)^2 B(v, u) = \sum_v x(v)^2 = ||x||^2,$$

where the inequality holds because $x \mapsto x^2$ is convex and $B$ is doubly stochastic. Since $x \mapsto x^2$ is *strictly* convex, the inequality is tight if and only if $x$ is constant on $\{v : B(v, u) > 0\}$ for every $u$—that is, for every $u$, $v$, and $v'$, if $B(v, u) > 0$ and $B(v', u) > 0$, then $x(v) = x(v')$. This gives the first part of the lemma. Note that this also shows that $||B$ is in fact a subspace. The second part of the lemma holds because $\perp B$ is defined as the orthogonal complement of $||B$. $\square$

We also measure how much $B$ mixes on its left (and right) connected components by $\lambda(\perp B)$ (and $\lambda(B\perp)$ respectively). Formally, define $\lambda(\perp B)$ and $\lambda(B\perp)$ to be the operator norm of $B$ restricted to $\perp B$ and $B\perp$—that is

$$\lambda(\perp B) := \max_{x \in \perp B : ||x||_2 = 1} ||xB||_2 \quad \text{and} \quad \lambda(B\perp) := \max_{x \in B\perp : ||x||_2 = 1} ||Bx||_2 .$$

We need the following bound on $\lambda(\perp B)$ and $\lambda(B\perp)$, which is analogous to the bound on the second eigenvalue of a connected undirected graph [16].

**Proposition 4.** *Let $B$ be a length-$n$ width-$w$ regular branching program. Then $\lambda(\perp B) \leq \lambda_w := 1 - 1/(16w^2)$ and likewise for $\lambda(B\perp)$.*

This bound is tight up to the constant factor 16. Crucially, this bound does not depend on $n$. De [4, Lemma 5.11] obtains the weaker bound $\lambda(\perp B) \leq 1 - w^{-3w}$.

*Proof.* For now, assume that $B$ is connected. Choose $x \in \perp B$ with $||x||_2 = 1$ and $||xB||_2 = \lambda(\perp B)$. Let $B = B_1 \circ B_2 \circ \cdots \circ B_n$. Define $x_0 = x$ and $x_i = x_{i-1}B_i$ for $i \in \{1, \cdots, n\}$. Then

$$\begin{aligned}
1 - \lambda(\perp B)^2 &= ||x||_2^2 - ||xB||_2^2 \\
&= ||x_0||_2^2 - ||x_n||_2^2 \\
&= \sum_{i\in\{1,\cdots,n\}} \left(||x_{i-1}||_2^2 - ||x_i||_2^2\right) \\
&= \sum_{i\in\{1,\cdots,n\}} \left(||x_{i-1}||_2^2 - ||x_{i-1}B_i||_2^2\right) \\
&= \sum_{i\in\{1,\cdots,n\}} \left(\frac{1}{2}\left(\sum_{u\in[w]} x_{i-1}(u)^2 + \sum_{v\in[w]} x_{i-1}(v)^2\right) - x_{i-1}B_iB_i^*x_{i-1}^*\right) \\
&= \sum_{i\in\{1,\cdots,n\}} \left(\begin{array}{l} \frac{1}{2}\sum_{u\in[w]} x_{i-1}(u)^2 \left(\sum_{v\in[w]}(B_iB_i^*)(u,v)\right) \\ +\frac{1}{2}\sum_{v\in[w]} x_{i-1}(v)^2 \left(\sum_{u\in[w]}(B_iB_i^*)(u,v)\right) \\ -\sum_{u,v\in[w]} x_{i-1}(u)(B_iB_i^*)(u,v)x_{i-1}(v) \end{array}\right) \\
&\quad \text{(since } B_iB_i^* \text{ is doubly stochastic)} \\
&= \sum_{i\in\{1,\cdots,n\}} \left(\sum_{u,v\in[w]} (B_iB_i^*)(u,v)\left(\frac{1}{2}x_{i-1}(u)^2 + \frac{1}{2}x_{i-1}(v)^2 - x_{i-1}(u)x_{i-1}(v)\right)\right) \\
&= \frac{1}{2} \sum_{i\in\{1,\cdots,n\}} \sum_{u,v\in[w]} (B_iB_i^*)(u,v)(x_{i-1}(u) - x_{i-1}(v))^2 \\
&\geq \frac{1}{8} \sum_{i\in\{1,\cdots,n\}} \sum_{u,v\in[w]:(B_iB_i^*)(u,v)>0} (x_{i-1}(u) - x_{i-1}(v))^2,
\end{aligned}$$

where the inequality follows from the fact that $B_i(u, v)$ is either 0, 1/2, or 1 for all $i$, $u$, and $v$.

Now we define a game. The game keeps track of a collection of closed intervals on the real line. Initially there are $w$ intervals (of zero length) of the form $[x_0(u), x_0(u)]$. At each step in the game we add an interval to the collection and, if that interval intersects other intervals, the connected intervals are merged into one large interval. We add all the intervals of the form $[x_{i-1}(u), x_{i-1}(v)]$ with $(B_iB_i^*)(u, v) > 0$ in order of increasing $i$—that is, if $i < i'$, then $[x_{i-1}(u), x_{i-1}(v)]$ is added before $[x_{i'-1}(u'), x_{i'-1}(v')]$.

We only consider the steps where the total length of the collection of intervals increases. Let $[x_{i-1}(u), x_{i-1}(v)]$ with $(i, u, v) \in S$ be the relevant intervals. Let $\alpha$ be the total length of the intervals at the end of the game. Then

$$\sum_{(i,u,v)\in S} |x_{i-1}(u) - x_{i-1}(v)| \geq \alpha.$$

Note that

$$1 - \lambda(\perp B)^2 \geq \frac{1}{8} \sum_{(i,u,v)\in S} (x_{i-1}(u) - x_{i-1}(v))^2 \geq \frac{1}{8|S|} \left( \sum_{(i,u,v)\in S} |x_{i-1}(u) - x_{i-1}(v)| \right)^2 \geq \frac{\alpha^2}{8|S|},$$

where the middle inequality follows from the fact that $||v||_2^2 \geq \frac{1}{m} ||v||_1^2$ for all $v \in \mathbb{R}^m$. So

$$1 - \lambda(\perp B) = \frac{1 - \lambda(\perp B)^2}{1 + \lambda(\perp B)} \geq \frac{1 - \lambda(\perp B)^2}{2} \geq \frac{\alpha^2}{16|S|}.$$

Now we must upper bound $|S|$ and lower bound $\alpha$. We make some observations about the game:

(i) At each step, the number of intervals either stays the same or decreases, as the added interval merges with existing intervals: Suppose we add the interval $[x_{i-1}(u), x_{i-1}(v)]$. Then the point $x_{i-1}(u)$ is already contained in an interval, so the added interval merges with that interval: If $i = 1$, then $x_{i-1}(u)$ is contained in one of the initial intervals. Otherwise, note that $x_{i-1}(u) = (x_{i-2}B_{i-1})(u) = (x_{i-2}(a) + x_{i-2}(b))/2$ for some $a$ and $b$; thus $x_{i-1}(u)$ is the midpoint of the interval $[x_{i-2}(a), x_{i-2}(b)]$, which has already been added.

(ii) At each step, if the total length of the collection of intervals increases, then the number of intervals decreases: Suppose we add the interval $[x_{i-1}(u), x_{i-1}(v)]$. As in (i), both endpoints $x_{i-1}(u)$ and $x_{i-1}(v)$ are contained in existing intervals. If both endpoints are in the same interval, the total length remains unchanged. If instead the endpoints are in different intervals, then those intervals are merged with the new interval and the number of intervals will decrease. Also note that the total length of the intervals can increase by at most the length of the new interval.

(iii) Since $B$ is connected, at the end of the game there is only one interval: Let $(u, v)$ be an edge in the graph $G(B)$ defined in Lemma 3. It suffices to show that $x_0(u)$ and $x_0(v)$ are in the same interval at the end of the game, as, by transitivity and the connectedness of $G(B)$, this implies that all the intervals have merged.

We have $(BB^*)(u, v) > 0$, whence there exist sequences $u_0, u_1, \cdots, u_n$ and $v_0, v_1, \cdots, v_n$ such that $u_0 = u$, $v_0 = v$, $u_n = v_n$, and, for each $i$, $B_i(u_{i-1}, u_i) > 0$ and $B_i(v_{i-1}, v_i) > 0$. (To see this, note that $(BB^*)(u, v) > 0$ implies that independent random walks in the branching program $B$ starting at $u$ and $v$ have a nonzero probability of colliding. The two sequences are two such colliding walks.)

For each $i$, there is some $u'_{i-1}$ such that $x_i(u_i) = (x_{i-1}(u_{i-1}) + x_{i-1}(u'_{i-1}))/2$ and $B_i(u'_{i-1}, u_i) > 0$. Since $x_i(u_i), x_{i-1}(u_{i-1}) \in [x_{i-1}(u_{i-1}), x_{i-1}(u'_{i-1})]$ and $(B_iB_i^*)(u_{i-1}, u'_{i-1}) > 0$, both $x_i(u_i)$ and $x_{i-1}(u_{i-1})$ are in the same interval at the end of the game. Likewise, for each $i$, both $x_i(v_i)$ and $x_{i-1}(v_{i-1})$ are covered by one interval. By transitivity, $x_0(u)$ and $x_0(v)$ are in the same interval at the end of the game.

(iv) The one final interval contains 0: Note that, since $x \in \perp B$, by Lemma 3, $\sum_u x(u) = 0$. Thus there exist $u$ and $v$ with $x_0(u) \leq 0 \leq x_0(v)$. But the final interval must contain the initial points $x_0(u)$ and $x_0(v)$, which implies that it contains 0.

By observation (ii), there are at most $w-1$ steps where the total length of the collection of intervals increases, as the number of intervals can only decrease $w-1$ times. Thus $|S| \leq w-1$.

Now it only remains to lower bound $\alpha$. By observation (iii), there is only one interval at the end of the game. This one final interval must contain all the initial intervals and, by observation (iv), 0. Thus

$$\alpha \geq \max_u |x_0(u) - 0| = ||x||_\infty \geq \frac{1}{\sqrt{w}} ||x||_2 = \frac{1}{\sqrt{w}}.$$

This gives the required result, assuming that $B$ is connected.

If $B$ is not connected, we must change the proof of observations (iii) and (iv). We can still show that every connected component of $G(B)$ is covered by one interval—that is, for any connected component $C$ of $G(B)$, at the end of the game there is one interval $I_C$ that contains all the points $\{x_0(u) : u \in C\}$. However, we can also show that $0 \in I_C$, as, by Lemma 3, $\sum_{u \in C} x_0(u) = 0$. Since all the intervals $I_C$ intersect at 0, they have been merged into one interval. Thus there is still only one final interval and the rest of the proof proceeds as before.

□

Let $B$ and $E$ be a matrices. We say that $E$ is a **natural error** of $B$ if $E(u,v) \neq 0$ implies that $B(u,v) > 0$ and $B + E$ is a stochastic matrix. If in addition $B + E$ is doubly stochastic, then we say that $E$ is a **doubly natural error** of $B$. Equivalently we say that $\tilde{B} = B + E$ is a **(doubly) natural approximation** of $B$. Note that, if $B$ is a program and $X$ an appropriate random variable, then $B[X]$ is a natural approximation to $B$. And, if, in addition, $B$ is a permutation program, then $B[X]$ is a doubly natural approximation to $B$.

## 2.6 Frobenius Norm

Let $B$ be an $n \times n$ matrix. Then the **Frobenius norm** of $B$ is defined by

$$||B||_{\mathrm{Fr}}^2 := \sum_{i,j} |B(i,j)|^2 = \mathrm{trace}(B^* B) = \sum_\lambda |\lambda|^2,$$

where the last sum is taken over the singular values of $B$. Note that the Frobenius norm is just the Euclidean norm on the vector space of matrices, which makes it clear that it is indeed a norm and that it is convex.

We use the Frobenius norm throughout this paper, so we will establish some basic facts about it.

**Lemma 5.** *Let $B$ be a stochastic $w \times w$ matrix. Then*

$$1 \leq ||B||_{Fr} \leq \sqrt{w}.$$

*Moreover, $||B||_{Fr} = 1$ if and only if $B = J$—that is, $B$ takes any distribution to uniform. And $||B||_{Fr} = \sqrt{w}$ if and only if $B$ has entries in $\{0, 1\}$.*

*Proof.* Let $b_u$ be the $u^{\text{th}}$ row of $B$. Then

$$||B||_{\mathrm{Fr}}^2 = \sum_{u \in [w]} ||b_u||_2^2 = \sum_i \mathrm{CP}(b_u),$$

where $\mathrm{CP}(b_u)$ is the collision probability of two independent walks starting at node $u$ when viewing $B$ as a stochastic process. We have

$$1/w \le \mathrm{CP}(b_u) \le 1,$$

where each inequality is tight if and only if $b_u$ is uniform or deterministic respectively. Applying this inequality to the sum gives the desired bounds, which can only be tight if the inequality is tight for each $u$. □

We view the Frobenius norm of a stochastic matrix $B$ as a measure of the "randomness" of $B$: We can view $B$ as a Markov chain, where $B(u,v)$ is the probability of ending at node $v$, when starting at node $u$. If $||B||_{\mathrm{Fr}} = 1$, then $B$ has maximal randomness, as it takes any start node to a uniformly random node. If $||B||_{\mathrm{Fr}} = \sqrt{w}$, then $B$ has minimal randomness, as it represents a deterministic stochastic process—every start node leads to only one final node. Note that $||B||_{\mathrm{Fr}}^2$ is the sum over all start nodes of the collision probability of two independent random steps.

The following are useful inequalities for our analysis.

**Lemma 6.** *Let $A$ be a doubly stochastic $w \times w$ matrix and $B$ another $w \times w$ matrix. Then*

$$||AB||_{Fr} \le ||B||_{Fr} \quad \text{and} \quad ||BA||_{Fr} \le ||B||_{Fr}.$$

Lemma 6 implies that the composition of two regular branching programs is at least as random as each individual program.

*Proof.* Let $a_u$ and $b_u$ be the $u^{\text{th}}$ rows of $A$ and $B$ respectively. Then

$$||AB||_{\mathrm{Fr}}^2 = \sum_u ||a_u B||_2^2 = \sum_u \left|\left| \sum_v a_u(v) b_v \right|\right|_2^2 \le \sum_{u,v} a_u(v) \, ||b_v||_2^2 = \sum_v ||b_v||_2^2 = ||B||_{\mathrm{Fr}}^2 .$$

□

**Lemma 7.** *Let $A$ and $B$ be $w \times w$ matrices. Then*

$$||AB||_{Fr} \le ||A||_{Fr} \, ||B||_{Fr} .$$

*Proof.* We have

$$||AB||_{\mathrm{Fr}}^2 = \sum_{u,v} \left( \sum_s A(u,s) B(s,v) \right)^2 \le \sum_{u,v} \left( \sum_s A(u,s)^2 \right) \left( \sum_s B(s,v)^2 \right) = ||A||_{\mathrm{Fr}}^2 \, ||B||_{\mathrm{Fr}}^2 ,$$

where the inequality follows from Cauchy-Schwarz. □

We can generalise Lemmas 6 and 7 to $||AB||_{\mathrm{Fr}} \le ||A||_2 \, ||B||_{\mathrm{Fr}}$ and $||AB||_{\mathrm{Fr}} \le ||A||_{\mathrm{Fr}} \, ||B||_2$, where $||B||_2 = \max_x \max\{||xB||_2 , ||Bx||_2\} / \, ||x||_2$ is the spectral norm. Note that $||B||_2 \le ||B||_{\mathrm{Fr}}$.

## 3 Results

Before we prove our results, we sketch the original analysis of the INW generator in order to clarify how we improve on it.

**Proposition 8.** *The INW generator fools width-$w$ length-$n$ branching programs with error $O(n \cdot w \cdot \lambda_H)$, where $\lambda_H$ is the second eigenvalue of the expander graphs being used.*

*Proof Sketch.* Let $\varepsilon_i$ be a bound on the error for width-$w$ branching programs of $G_i$, the INW generator of length $2^i$, when using expanders with second eigenvalue bounded by $\lambda_H$—that is, for any width-$w$ length-$2^i$ branching program $B$, we have

$$||B[G_i(U_{i\cdot d+1})] - B||_{\mathrm{Fr}} \le \varepsilon_i,$$

where $i \cdot d + 1$ is the seed length of $G_i$ and $U_{i\cdot d+1}$ is uniform on $\{0,1\}^{i\cdot d+1}$. We simply need to prove that $\varepsilon_{\log n} = O(n \cdot w \cdot \lambda_H)$.

Clearly $\varepsilon_0 = 0$. This forms the base case of our induction. Now we bound $\varepsilon_{i+1}$ in terms of $\varepsilon_i$: Let $B_0$ and $B_1$ be two width-$w$ length-$2^i$ branching programs and $B = B_0 \circ B_1$ a width-$w$ length-$2^{i+1}$ branching program. Let $X$ be uniform on $\{0,1\}^{i\cdot d+1}$. Then

$$\begin{aligned}\left|\left|B[G_{i+1}(U_{(i+1)d})] - B\right|\right|_{\mathrm{Fr}} =& \,||\mathbb{E}\left[B_0[G_i(X)]B_1[G_i(H(X,U_d))]\right] - B_0B_1||_{\mathrm{Fr}}\\ \le& \,||\mathbb{E}\left[B_0[G_i(X)]B_1[G_i(H(X,U_d))]\right] - B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})]||_{\mathrm{Fr}}\\ &+ ||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}}\\ \le& \,w\lambda_H + ||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}},\end{aligned}$$

where the last inequality follows from standard facts about expander graphs. Now it remains to bound $||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}}$. We have

$$\begin{aligned}||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}} \le& \,||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1[G_i(U_{i\cdot d+1})]||_{\mathrm{Fr}}\\ &+ ||B_0B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}}\\ \le& \,||B_0[G_i(U_{i\cdot d+1})] - B_0||_{\mathrm{Fr}} + ||B_1[G_i(U_{i\cdot d+1})] - B_1||_{\mathrm{Fr}}\\ \le& \,\varepsilon_i + \varepsilon_i.\end{aligned}$$

Thus we have the recurrence

$$\varepsilon_{i+1} \le 2\varepsilon_i + w\lambda_H,$$

which solves to $\varepsilon_i = O(2^i w\lambda_H)$, as required. □

One way to improve on the original analysis of the INW generator is to reduce the $w\lambda_H$ term added with each recursion. For permutation programs, this term can be reduced to $\lambda_H$ [14, 4]. Changing the generator to more aggressively recycle randomness can also improve this term [10].

However, it is more fruitful to improve the bound

$$||B_0[G_i(U_{i\cdot d+1})]B_1[G_i(U_{i\cdot d+1})] - B_0B_1||_{\mathrm{Fr}} \le ||B_0[G_i(U_{i\cdot d+1})] - B_0||_{\mathrm{Fr}} + ||B_1[G_i(U_{i\cdot d+1})] - B_1||_{\mathrm{Fr}}.$$

This is what we do to achieve our results. This bound comes from the inequality

$$\left|\left|\tilde{A}B - AB\right|\right|_{\mathrm{Fr}} \leq \left|\left|\tilde{A} - A\right|\right|_{\mathrm{Fr}}.$$

We sharpen this inequality in two ways in Sections 3.1 and 3.2. These sharpened bounds are assembled to give our main result in Section 3.3.

## 3.1 "Introducing Error Requires Introducing Randomness"

Consider $\tilde{A}B$ versus $AB$, where $A$ and $B$ are programs and $\tilde{A}$ is a natural approximation to $A$. Suppose that $A$ does not contribute randomness to the product $AB$—that is, $||AB||_{\mathrm{Fr}} = ||B||_{\mathrm{Fr}}$. Then we can show that $A$ does not contribute any error to the product either—that is, $\tilde{A}B = AB$. Formally, we have the following.

**Proposition 9.** *Let $A$ and $B$ be doubly stochastic matrices and let $E$ be a natural error of $A$. Suppose that $||AB||_{Fr} = ||B||_{Fr}$. Then $EB = 0$.*

The assumption $||AB||_{\mathrm{Fr}} = ||B||_{\mathrm{Fr}}$ implies that $A$ is not mixing $B$. The only way $A$ cannot be mixing is if it is acting on a uniform distribution. This means that $A$ is irrelevant—that is, if $A$ and $B$ are programs, $A[x]B = A[y]B$ for all $x$ and $y$.

*Proof.* Let $b_j$ be the $j^{\mathrm{th}}$ column of $B$. Then $||Ab_j||_2 = ||b_j||_2$ for all $j$. So $b_j \in A||$ for all $j$. By Lemma 3, this means that, if $(A^*A)(u,v) > 0$, then $b_j(u) = b_j(v)$.

Let $a_i$ and $e_i$ be the $i^{\mathrm{th}}$ rows of $A$ and $E$ respectively. Fix $i$ and $j$. Suppose that $e_i(u) \neq 0$ and $e_i(v) \neq 0$. Then $a_i(u) > 0$ and $a_i(v) > 0$. Thus $(A^*A)(u,v) > 0$ and $b_j(u) = b_j(v)$. Since $\sum_u e_i(u) = 0$, we have

$$(EB)(i,j) = e_i b_j = \sum_u e_i(u) b_j(u) = b_j(u^*) \sum_u e_i(u) = 0.$$

Thus $EB = 0$. $\square$

We have the following quantitative version of Proposition 9.

**Proposition 10.** *Let $A$ and $B$ be doubly stochastic matrices and let $E$ be a natural error of $A$. Suppose that $||AB||_{Fr}^2 = ||B||_{Fr}^2 - \gamma$. Then*

$$||EB||_{Fr} \leq ||E||_{Fr} \sqrt{\frac{\gamma}{1 - \lambda(A\perp)^2}}.$$

*Proof.* Let $b_j$ be the $j^{\mathrm{th}}$ column of $B$. Decompose $b_j$ into $b_j^{||} \in A||$ and $b_j^{\perp} \in A\perp$. Let $B = B^{||} + B^{\perp}$, where $b_j^{||}$ and $b_j^{\perp}$ are the $j^{\mathrm{th}}$ columns of $B^{||}$ and $B^{\perp}$ respectively. Here $B^{||}$ is the component of $B$ that is indifferent to $A$, as it is uniform on the connected components of $A$ and $B^{\perp}$ is orthogonal to $B^{||}$ and, as we will see, small.

Now

$$||B||_{\mathrm{Fr}}^2 = \sum_j \left( \left|\left| b_j^{||} \right|\right|_2^2 + \left|\left| b_j^{\perp} \right|\right|_2^2 \right) \quad \text{and} \quad ||AB||_{\mathrm{Fr}}^2 = \sum_j \left( \left|\left| b_j^{||} \right|\right|_2^2 + \left|\left| Ab_j^{\perp} \right|\right|_2^2 \right).$$

So

$$\gamma = ||B||_{\mathrm{Fr}}^2 - ||AB||_{\mathrm{Fr}}^2 = \sum_j \left( \left|\left| b_j^{\perp} \right|\right|_2^2 - \left|\left| Ab_j^{\perp} \right|\right|_2^2 \right) \geq (1 - \lambda(A\perp)^2) \sum_j \left|\left| b_j^{\perp} \right|\right|_2^2,$$

whence

$$\left|\left| B^{\perp} \right|\right|_{\mathrm{Fr}}^2 = \sum_j \left|\left| b_j^{\perp} \right|\right|_2^2 \leq \frac{\gamma}{1 - \lambda(A\perp)^2}.$$

By Proposition 9, $EB^{||} = 0$. Now, by Lemma 7,

$$||EB||_{\mathrm{Fr}} = \left|\left| EB^{\perp} \right|\right|_{\mathrm{Fr}} \leq ||E||_{\mathrm{Fr}} \left|\left| B^{\perp} \right|\right|_{\mathrm{Fr}} \leq ||E||_{\mathrm{Fr}} \sqrt{\frac{\gamma}{1 - \lambda(A\perp)^2}}.$$

□

By taking transposes, we immediately obtain the following.

**Corollary 11.** *Let $A$ and $B$ be doubly stochastic matrices and let $E$ be a doubly natural error of $B$. Suppose that $||AB||_{Fr}^2 = ||A||_{Fr}^2 - \gamma$. Then*

$$||AE||_{Fr} \leq ||E||_{Fr} \sqrt{\frac{\gamma}{1 - \lambda(\perp B)^2}}.$$

Note that Corollary 11 requires $E$ to be a *doubly* natural error. This is important, as we need to assume that $B$ is a permutation branching program in order to guarantee that the error from using a pseudorandom generator is doubly natural. Proposition 10 only requires a natural error, which does not require this assumption. This prevents us from analysing non-permutation branching programs.

## 3.2 "Mixing Kills Error"

Again consider $\tilde{A}B$ versus $AB$. Suppose for now that $B$ is connected and thus $||B - J||_2 = \lambda(B) < 1$. Then $B$ "mixes" the error in $A$, thereby reducing it. Then we have

$$\left|\left| \tilde{A}B - AB \right|\right|_{\mathrm{Fr}} \leq \left|\left| (\tilde{A} - A)J \right|\right|_{\mathrm{Fr}} + \left|\left| (\tilde{A} - A)(B - J) \right|\right|_{\mathrm{Fr}} \leq 0 + \left|\left| \tilde{A} - A \right|\right|_{\mathrm{Fr}} \lambda(B).$$

This is a modest improvement over the bound $\left|\left| \tilde{A}B - AB \right|\right|_{\mathrm{Fr}} \leq \left|\left| \tilde{A} - A \right|\right|_{\mathrm{Fr}}$, but it can be powerful when applied repeatedly.

Consider $\tilde{B}_1\tilde{B}_2 \cdots \tilde{B}_k$ versus $B_1B_2 \cdots B_k$. We have

$$\left|\left| \tilde{B}_1\tilde{B}_2 \cdots \tilde{B}_k - B_1B_2 \cdots B_k \right|\right| \leq \sum_i \left|\left| \tilde{B}_i - B_i \right|\right|.$$

However, assuming that each $B_i$ is connected, we have

$$\begin{aligned}\left|\left|\tilde{B}_1\tilde{B}_2\cdots\tilde{B}_k - B_1B_2\cdots B_k\right|\right| &\leq \sum_{i\in\{1,\cdots,k\}} \left|\left|\tilde{B}_1\tilde{B}_2\cdots\tilde{B}_{i-1}(\tilde{B}_i - B_i)B_{i+1}\cdots B_k\right|\right| \\ &\leq \sum_{i\in\{1,\cdots,k\}} \left|\left|\tilde{B}_1\tilde{B}_2\cdots\tilde{B}_{i-1}\right|\right| \left|\left|\tilde{B}_i - B_i\right|\right| \lambda(B_{i+1}\cdots B_k) \\ &\leq \sum_{i\in\{1,\cdots,k\}} \left|\left|\tilde{B}_i - B_i\right|\right| \lambda^{k-i} \\ &\leq \frac{1}{1-\lambda}\max_i \left|\left|\tilde{B}_i - B_i\right|\right|,\end{aligned}$$

where $\lambda$ is an upper bound on $\lambda(B_i)$. The crucial difference is that the sum is replaced by a maximum and thus this bound does not depend on $k$.

We can remove the assumption that each $B_i$ is connected and obtain the following result. We also add in expander products.

**Theorem 12** (Key Convergence Lemma). *Let $B_0, \cdots, B_k$ be width-$w$ regular branching programs and $\tilde{B}_0, \cdots, \tilde{B}_k$ programs with $\left|\left|\tilde{B}_i - B_i\right|\right|_{Fr} \leq \delta$ for $i \in \{1, \cdots, k\}$. Let $H_1, \cdots, H_k$ be expander graphs of the appropriate size and with second eigenvalue $\lambda_H$. Then*

$$\begin{aligned}&\left|\left|\left(\cdots\left(\left(\tilde{B}_0 \circ_{H_1} \tilde{B}_1\right) \circ_{H_2} \tilde{B}_2\right)\cdots\right) \circ_{H_k} \tilde{B}_k - B_0B_1\cdots B_k\right|\right|_{Fr} \\ &\leq \left(\sqrt{w}\lambda_H + \delta\right)\cdot\sqrt{w}\cdot w!\cdot\left(\frac{3}{1-\lambda_w}\right)^w + \left|\left|\tilde{B}_0 - B_0\right|\right|_{Fr} = (\lambda_H + \delta)\cdot 2^{O(w\log w)} + \left|\left|\tilde{B}_0 - B_0\right|\right|_{Fr}.\end{aligned}$$

The crucial aspect of Theorem 12 is that the bound does not depend on $k$.

Theorem 12 follows from Proposition 13. First we give a proof sketch.

*Proof Sketch.* We wish to reduce the problem to the case where each $B_i$ is connected. We also use induction on the width $w$. So suppose that the theorem holds for all programs of width at most $w-1$.

Now we prove the result for increasingly more general cases.

(a) Suppose $B_1B_2\cdots B_k$ is disconnected. Then we can simply apply the induction hypothesis on each connected component and obtain the desired bound.

(b) Suppose $B_1B_2\cdots B_k$ is "minimally connected"—that is, $B_1B_2\cdots B_{k-1}$ is disconnected and $B_1B_2\cdots B_k$ is connected. Then we can use case (a) to bound the error in $B_1B_2\cdots B_{k-1}$ and a basic analysis to bound the error from the last part.

(c) In general, partition $B_1B_2\cdots B_k$ into minimally connected blocks of the form $B_{k_i+1}B_{k_i+2}\cdots B_{k_{i+1}}$. Cases (a) and (b) cover the error added by each block. Now, since each block is connected, we can use the same analysis as when every $B_i$ is connected.

□

Proposition 13 is simply Theorem 12 restricted to a single starting vertex (so we analyse vectors, rather than matrices) with several technical conditions needed for the induction.

**Proposition 13.** *Suppose that all of the following hold.*

- *For $i \in \{1, \cdots, k\}$, $B_i$ is a length-$n_i$ width-$w$ regular branching program and $\tilde{B}_i$ is a length-$\tilde{n}_i$ width-$w$ program, with $\left|\left|\tilde{B}_i - B_i\right|\right|_{Fr} \leq \delta$.*
- *$b_0 : \{0,1\}^{n_0} \to [w]$ and $\tilde{b}_0 : \{0,1\}^{\tilde{n}_0} \to [w]$ are functions, the output of which we equate with a row vector in $\mathbb{R}^w$.*
- *$Y_1, \cdots, Y_k$ are independent uniform random variables over $\{0,1\}^d$.*
- *$X_0$ is uniform over $\tilde{S} \subset \{0,1\}^{\tilde{n}_0}$ and independent from $Y_1, \cdots, Y_k$.*
- *For $i \in \{1, \cdots, k\}$, we have $X_i = H_i(Y_i, (Y_{i-1}, \cdots, Y_1, X_0)) \in \{0,1\}^{\tilde{n}_i}$, where $H_i$ is a $2^d$-regular $\lambda_H$-expander. Note that this implies that $\tilde{n}_i = \tilde{n}_0 + (i-1)d$ for $i \in \{1, \cdots, k\}$.*
- *$U_S$ is uniform over $S \subset \{0,1\}^{n_0}$.*

*Then*

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right]B_1\cdots B_k\right|\right|_2 \leq f\left(w, \lambda_w, \lambda_H, \delta, \frac{|\tilde{S}|}{2^{\tilde{n}_0}}\right),$$

*where*

$$f(w, \lambda_w, \lambda_H, \delta, \mu) = \frac{1}{\mu}\left(\sqrt{w}\lambda_H + \delta\right) w! \left(\frac{3}{1-\lambda_w}\right)^w.$$

The triangle inequality gives

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_k]\right] - \mathbb{E}\left[b_0[U_S]\right]B_1\cdots B_k\right|\right|_2 \leq f(w, \lambda_w, \lambda_H, \delta, \mu_{w'}) + \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right|\right|_2.$$

*Proof.* We proceed by induction on $w$. Clearly the proposition holds for $w = 1$. Now assume that it holds for width strictly less than $w$.

In all of the following lemmas we assume the hypotheses of Proposition 13 and the induction hypothesis.

Denote $\mu(S) = |S|/2^{n_0}$ and $\mu(\tilde{S}) = |\tilde{S}|/2^{\tilde{n}_0}$.

**Lemma 14.** *Suppose that $B_1 \cdots B_k$ is disconnected. Then*

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots \tilde{B}_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right]B_1\cdots B_k\right|\right|_2 \leq wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right).$$

*Proof.* We partition the program (and $\tilde{S}$) into several smaller programs and apply the induction hypothesis.

Let $C_1, \cdots C_t$ be the left connected components of $B_1 \cdots B_k$. For $i \in \{1, \cdots, t\}$, define

$$S_i = \{x \in S : b_0[x] \in C_i\} \quad \text{and} \quad \tilde{S}_i = \{x \in \tilde{S} : \tilde{b}_0[x] \in C_i\}.$$

Denote $\mu(S_i) = |S_i|/2^{n_0}$ and $\mu(\tilde{S}_i) = |\tilde{S}_i|/2^{\tilde{n}_0}$. We can apply the induction hypothesis to each connected component—that is, for all $i$,

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k] \mid X_0 \in \tilde{S}_i\right] - \mathbb{E}\left[\tilde{b}_0[X_0] \mid X_0 \in \tilde{S}_i\right] B_1 \cdots B_k\right|\right|_2 \le f\left(|C_i|, \lambda_w, \lambda_H, \delta, \mu(\tilde{S}_i)\right).$$

Note that

$$\begin{aligned}\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k]\right] &= \sum_i \mathbb{P}\left[X_0 \in \tilde{S}_i\right]\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k] \mid X_0 \in \tilde{S}_i\right] \quad \text{and} \\ \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1 \cdots B_k &= \sum_i \mathbb{P}\left[X_0 \in \tilde{S}_i\right]\mathbb{E}\left[\tilde{b}_0[X_0] \mid X_0 \in \tilde{S}_i\right] B_1 \cdots B_k.\end{aligned}$$

Thus

$$\begin{aligned}&\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1 \cdots B_k\right|\right|_2 \\ &= \left|\left|\sum_i \mathbb{P}\left[X_0 \in \tilde{S}_i\right]\left(\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k] \mid X_0 \in \tilde{S}_i\right] - \mathbb{E}\left[\tilde{b}_0[X_0] \mid X_0 \in \tilde{S}_i\right] B_1 \cdots B_k\right)\right|\right|_2 \\ &\le \sum_i \left|\left|\mathbb{P}\left[X_0 \in \tilde{S}_i\right]\left(\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k] \mid X_0 \in \tilde{S}_i\right] - \mathbb{E}\left[\tilde{b}_0[X_0] \mid X_0 \in \tilde{S}_i\right] B_1 \cdots B_k\right)\right|\right|_2 \\ &\le \sum_i \mathbb{P}\left[X_0 \in \tilde{S}_i\right] f\left(|C_i|, \lambda_w, \lambda_H, \delta, \mu(\tilde{S}_i)\right) \\ &\le \sum_i \frac{\mu(\tilde{S}_i)}{\mu(\tilde{S})} f\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S}_i)\right) \\ &= tf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) \\ &\le wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right),\end{aligned}$$

where the last equality follows from the fact that $f(w-1, \lambda_w, \lambda_H, \delta, \mu) = (1/\mu) \cdot g(w-1, \lambda_w, \lambda_H, \delta)$ for some $g$. □

Lemma 15 is a simple one-step analysis. The proof is omitted, as it is a simple calculation based on a standard result [16].

**Lemma 15.**

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\right] - \mathbb{E}\left[b_0[U_S]\right] B_1\right|\right|_2 \le \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right|\right|_2$$

Now Lemma 14 covers the case that the whole program is disconnected and Lemma 15 gives an analysis of one step. We can combine these two results to analyse the minimally connected case.

**Lemma 16.** *Suppose that $B_1 \cdots B_k$ is connected, but $B_1 \cdots B_{k-1}$ is not. Then*

$$\begin{aligned}&\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k]\right] - \mathbb{E}\left[b_0[U_S]\right] B_1\cdots B_k\right|\right|_2 \\ &\leq \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) + \lambda_w \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right|\right|_2.\end{aligned}$$

*Proof.* By Lemma 14,

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1\cdots B_{k-1}\right|\right|_2 \leq wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right).$$

Now Lemma 15 gives

$$\begin{aligned}&\left|\left|\mathbb{E}\left[\left(\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right)\tilde{B}_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]B_1\cdots B_{k-1}\right] B_k\right|\right|_2 \\ &\leq \left|\left|\mathbb{E}\left[\left(\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right)\tilde{B}_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right] B_k\right|\right|_2 \\ &\quad + \left|\left|\left(\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1\cdots B_{k-1}\right) B_k\right|\right|_2 \\ &\leq \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1\cdots B_{k-1}\right|\right|_2 \\ &\leq \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right).\end{aligned}$$

Thus

$$\begin{aligned}&\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_k[X_k]\right] - \mathbb{E}\left[b_0[U_S]\right] B_1\cdots B_k\right|\right|_2 \\ &\leq \left|\left|\mathbb{E}\left[\left(\tilde{b}_0[X_0]\tilde{B}_1[X_1]\cdots\tilde{B}_{k-1}[X_{k-1}]\right)\tilde{B}_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]B_1\cdots B_{k-1}\right] B_k\right|\right|_2 \\ &\quad + \left|\left|\left(\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right) B_1\cdots B_k\right|\right|_2 \\ &\leq \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) + \lambda(\perp(B_1\cdots B_k)) \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right|\right|_2 \\ &\leq \frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) + \lambda_w \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\right] - \mathbb{E}\left[b_0[U_S]\right]\right|\right|_2,\end{aligned}$$

where the last inequality follows from Proposition 4. ☐

Now we partition $B_1 B_2 \cdots B_k$ into minimally connected blocks. Choose $k_0 \cdots k_t$ such that $k_0 = 0$ and, for all $i$, $B_{k_i+1} \cdots B_{k_{i+1}}$ is connected and $B_{k_i+1} \cdots B_{k_{i+1}-1}$ is not connected and $B_{k_t+1} \cdots B_k$ is also not connected.

Now we can apply the same analysis to the connected blocks as we did in the case where each individual $B_i$ was connected.

**Lemma 17.** *For all $i \in \{0, \cdots, t\}$,*

$$\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1] \cdots \tilde{B}_{k_i}[X_{k_i}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1 \cdots B_{k_i}\right|\right|_2 \leq \frac{\frac{\lambda_H \sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right)}{1-\lambda_w}$$

*Proof.* We apply Lemma 16 to each segment $B_{k_i+1} \cdots B_{k_{i+1}}$ and proceed by induction on $i$. Clearly the claim is true for $i = 0$. Suppose the claim is true for $i$. Then, by Lemma 16,

$$\begin{aligned}
&\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1] \cdots \tilde{B}_{k_{i+1}}[X_{k_{i+1}}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1 \cdots B_{k_{i+1}}\right|\right|_2 \\
&\leq \frac{\lambda_H \sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) \\
&\quad + \lambda_w \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]\tilde{B}_1[X_1] \cdots \tilde{B}_{k_i}[X_{k_i}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right] B_1 \cdots B_{k_i}\right|\right|_2 \\
&\leq \frac{\lambda_H \sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right) \\
&\quad + \lambda_w \frac{\frac{\lambda_H \sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right)}{1-\lambda_w} \\
&= \frac{\frac{\lambda_H \sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1, \lambda_w, \lambda_H, \delta, \mu(\tilde{S})\right)}{1-\lambda_w}.
\end{aligned}$$

□

All that remains is to deal with the last block that doesn't fit into the minimally connected components analysis.

Now we can combine Lemmas 17 and 14 to complete the induction.

$$\begin{aligned}
&\left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right]B_1\cdots B_k\right|\right|_2 \\
&\le \left|\left|\mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_k]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_{k_t}]\right]B_{k_t+1}\cdots B_k\right|\right|_2 \\
&\quad + \left|\left|\left(\mathbb{E}\left[\tilde{b}_0[X_0]B_1[X_1]\cdots B_k[X_{k_t}]\right] - \mathbb{E}\left[\tilde{b}_0[X_0]\right]B_1\cdots B_{k_t}\right)B_{k_t+1}\cdots B_k\right|\right|_2 \\
&\le wf\left(w-1,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right) \\
&\quad + \frac{\frac{\lambda_H\sqrt{w}}{\mu(\tilde{S})} + \delta + wf\left(w-1,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right)}{1-\lambda_w} \\
&\le \left(1+\frac{1}{1-\lambda_w}\right)wf\left(w-1,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right) + \frac{1}{\mu(\tilde{S})}\frac{\lambda_H\sqrt{w}+\delta}{1-\lambda_w} \\
&= \frac{1+\frac{1}{1-\lambda_w}}{\frac{3}{1-\lambda_w}}f\left(w,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right) + \frac{1}{\mu(\tilde{S})}\frac{\lambda_H\sqrt{w}+\delta}{1-\lambda_w} \\
&\le \frac{2}{3}f\left(w,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right) + \frac{1}{\mu(\tilde{S})}\frac{\lambda_H\sqrt{w}+\delta}{1-\lambda_w} \\
&\le f\left(w,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right),
\end{aligned}$$

as

$$\frac{1}{3}f\left(w,\lambda_w,\lambda_H,\delta,\mu(\tilde{S})\right) = \frac{1}{3}\frac{1}{\mu(\tilde{S})}\left(\sqrt{w}\lambda_H+\delta\right)w!\left(\frac{3}{1-\lambda_w}\right)^w \ge \frac{1}{\mu(\tilde{S})}\frac{\lambda_H\sqrt{w}+\delta}{1-\lambda_w}.$$

□

Note that the expander product is not associative and Theorem 12 only allows the expander product to be repeatedly applied on the right. This can be generalised. The following proposition shows how we can reduce arbitrary orders of expander products to the case where every product is on the right.

**Proposition 18.** *Let $B_0,\cdots,B_k,B'_1,\cdots,B'_k$ be width-$w$ permutation branching programs and $\tilde{B}_0,\cdots,\tilde{B}_k,\tilde{B}'_1,\cdots,\tilde{B}'_k$ permutation programs with $\left|\left|\tilde{B}_i - B_i\right|\right|_{Fr} \le \delta$ and $\left|\left|\tilde{B}'_i - B'_i\right|\right|_{Fr} \le \delta$ for $i\in\{1,\cdots,k\}$. Let $H_1,\cdots,H_k,H'_1,\cdots,H'_k$ be expander graphs of the appropriate size and with second eigenvalue $\lambda_H$. Then*

$$\begin{aligned}
&\left|\left|\tilde{B}'_k \circ_{H'_k}\left(\left(\cdots\left(\tilde{B}'_2\circ_{H'_2}\left(\left(\tilde{B}'_1\circ_{H'_1}\left(\tilde{B}_0\circ_{H_1}\tilde{B}_1\right)\right)\circ_{H_2}\tilde{B}_2\right)\right)\cdots\right)\circ_{H_k}\tilde{B}_k\right) - B'_k\cdots B'_1B_0B_1\cdots B_k\right|\right|_{Fr} \\
&\quad\le 2\left(\sqrt{2w}\lambda_H+\delta\right)\sqrt{2w}(2w)!\left(\frac{3}{1-\lambda_{2w}}\right)^{2w} + 2\left|\left|\tilde{B}_0 - B_0\right|\right|_{Fr} = (\lambda_H+\delta)2^{O(w\log w)} + 2\left|\left|\tilde{B}_0 - B_0\right|\right|_{Fr}.
\end{aligned}$$

To prove Proposition 18, we give a program that simulates the given order of products using only right expander products and twice the width. This is possible because a permutation branching program can be "run backwards:" Suppose we wish to simulate only left expander products. Consider the transpose of a sequence of left multiplications. For permutation programs, this transpose is still a permutation program. This is because each node has distinct labels on incoming edges. So we can use the same edge labelling to take a random walk backwards through the program. Non-permutation branching programs do not satisfy this property—they "lose entropy". So, while Theorem 12, can be applied to non-permutation branching programs, Proposition 18 cannot. This prevents us from generalising our result to regular branching programs.

*Proof.* We give a program $A$ that simulates $B$ and $B'$ with the desired order of expander products. Define, for $i \in \{1, \cdots, k\}$,

$$A_0[x] = \begin{pmatrix} B_0[x] & 0 \\ 0 & I \end{pmatrix} \quad A_{2i-1}[x] = \begin{pmatrix} B_i[x] & 0 \\ 0 & I \end{pmatrix} \quad A_{2i}[x] = \begin{pmatrix} I & 0 \\ 0 & B_i'[x]^* \end{pmatrix}$$

and

$$\tilde{A}_0[x] = \begin{pmatrix} \tilde{B}_0[x] & 0 \\ 0 & I \end{pmatrix} \quad \tilde{A}_{2i-1}[x] = \begin{pmatrix} \tilde{B}_i[x] & 0 \\ 0 & I \end{pmatrix} \quad \tilde{A}_{2i}[x] = \begin{pmatrix} I & 0 \\ 0 & \tilde{B}_i'[x]^* \end{pmatrix},$$

where $B_i'[x]^*$ is the transpose of $B_i'[x]$. This is where we use the assumption that everything is a permutation program—otherwise $A_i$ and $\tilde{A}_i$ would not be programs. Now, by Theorem 12,

$$\begin{aligned} &\left|\left|\left(\cdots\left(\left(\tilde{A}_0 \circ_{H_1} \tilde{A}_1\right) \circ_{H_1'} \tilde{A}_2\right)\cdots\right) \circ_{H_k'} \tilde{A}_{2k} - A_0 A_1 \cdots A_{2k}\right|\right|_{\text{Fr}} \\ &\le \left(\sqrt{2w}\lambda_H + \delta\right)\sqrt{2w}(2w)!\left(\frac{3}{1-\lambda_{2w}}\right)^{2w} + \left|\left|\tilde{A}_0 - A_0\right|\right|_{\text{Fr}} = (\lambda_H + \delta)2^{O(w \log w)} + \left|\left|\tilde{B}_0 - B_0\right|\right|_{\text{Fr}}. \end{aligned}$$

Now we show how $A$ simulates $B$. Define

$$\pi\begin{pmatrix} X & 0 \\ 0 & Y \end{pmatrix} = Y^* X.$$

Then

$$\begin{aligned} &\tilde{B}_k' \circ_{H_k'} \left(\left(\cdots\left(\tilde{B}_2' \circ_{H_2'} \left(\left(\tilde{B}_1' \circ_{H_1'} \left(\tilde{B}_0 \circ_{H_1} \tilde{B}_1\right)\right) \circ_{H_2} \tilde{B}_2\right)\right)\cdots\right) \circ_{H_k} \tilde{B}_k\right) \\ &\quad = \pi\left(\left(\cdots\left(\left(\tilde{A}_0 \circ_{H_1} \tilde{A}_1\right) \circ_{H_1'} \tilde{A}_2\right)\cdots\right) \circ_{H_k'} \tilde{A}_{2k}\right) \quad \text{and} \\ &B_k' \cdots B_1' B_0 B_1 \cdots B_k = \pi\left(A_0 A_1 \cdots A_{2k}\right). \end{aligned}$$

Now we are done, as

$$||\pi(X) - \pi(Y)||_{\text{Fr}} \le 2\,||X - Y||_{\text{Fr}}.$$

□

## 3.3 Putting it all Together

Now we combine the two bounds from Sections 3.1 and 3.2 to give our main result.

**Theorem 19** (Main Result)**.** *The INW generator with seed length $O\left(\left(w^4 \log w + \log(1/\varepsilon)\right) \cdot \log n\right)$ $\varepsilon$-fools width-$w$ length-$n$ permutation branching programs—that is, for any length-$n$ width-$w$ permutation branching program $B$, if $G_{\lceil \log n \rceil}$ is the INW generator with $\lambda_H = \varepsilon \cdot 2^{O(w^4 \log w)}$, then*

$$\left|\left|B\left[G_{\lceil \log n \rceil}\left(U_{O((w^4 \log w + \log(1/\varepsilon)) \cdot \log n)}\right)\right] - B\right|\right|_{Fr} \leq \varepsilon.$$

The general approach of the proof is to identify the the right places to use each of our two bounds.

- The "Introducing Error Requires Introducing Randomness" bound from Section 3.1 is used when little randomness is introduced. In particular, when

$$\max\left\{||A||_{\mathrm{Fr}}^2, ||B||_{\mathrm{Fr}}^2\right\} - ||AB||_{\mathrm{Fr}}^2 \leq \frac{1}{O(w^2)},$$

  we can show that

$$\left|\left|\tilde{A}\tilde{B} - AB\right|\right|_{\mathrm{Fr}} \leq \frac{1}{8} \max\left\{\left|\left|\tilde{A} - A\right|\right|_{\mathrm{Fr}}, \left|\left|\tilde{B} - B\right|\right|_{\mathrm{Fr}}\right\}.$$

- The "Mixing Kills Error" bound from Section 3.2 is somewhat more complicated, as the bound applies to sequences. Essentially it shows that

$$\left|\left|\prod_j \tilde{B}_j - \prod_j B_j\right|\right|_{\mathrm{Fr}} \leq 2^{O(w \log w)} \max_i \left|\left|\tilde{B}_i - B_i\right|\right|_{\mathrm{Fr}}.$$

  If randomness is added to every term in the sequence—that is,

$$\min_i ||B_i||_{\mathrm{Fr}}^2 \geq \left|\left|\prod_j B_j\right|\right|_{\mathrm{Fr}}^2 + \frac{1}{O(w^2)},$$

  then we can afford the $2^{O(w \log w)}$ blowup in error.

The $2^{O(w \log w)}$ blowup can happen only $O(w^3)$ times, as it only happens when $1/O(w^2)$ randomness is added (as measured by Frobenius norm squared) and there is only capacity for $w-1$ randomness to be added. Hence the final error blowup is $\left(2^{O(w \log w)}\right)^{O(w^3)}$. This means that the final error is bounded by $\lambda_H 2^{O(w^4 \log w)}$, where $\lambda_H$ is a bound on the second eigenvalue of the expanders used by the INW generator.

The proof proceeds by inductively proving that, for any permutation branching program $B$ with the appropriate parameters,

$$\left|\left|\tilde{B} - B\right|\right|_{\mathrm{Fr}} \leq \rho\left(||B||_{\mathrm{Fr}}^2\right),$$

where $\tilde{B}$ is the program obtained by applying the INW generator to $B$ and $\rho$ is some function. This bound effectively says

$$\text{“error of } B\text{”} \leq \rho\,(\text{“randomness of } B\text{”})\,.$$

This is a further formalisation of the “Introducing Error Requires Introducing Randomness” intuition from Section 3.1.

*Proof.* Let $B = B_1 \circ B_2 \circ \cdots \circ B_n$ be a width-$w$ length-$n$ permutation branching program. For simplicity, we assume that $n$ is a power of two. In general, we can round $n$ up to the next power of two and discard the extra pseudorandom bits.

Consider the binary recursion tree of the INW generator applied to $B$ with each node labelled by the program formed by its descendants. The leaves of the tree are $B_1, B_2, \cdots, B_n$. And an internal node is the concatenation of its two children. More formally, the $i^{\text{th}}$ node in the $j^{\text{th}}$ layer ($0 \leq j \leq \log n$ and $1 \leq i \leq n2^{-j}$) is denoted by $B_{i,j}$. If $j = 0$, $B_{i,j}$ is a leaf representing $B_i$. If $j > 0$, $B_{i,j}$ has left and right children $B_{2i-1,j-1}$ and $B_{2i,j-1}$ respectively and the program represented by $B_{i,j}$ is

$$B_{i,j} = B_{i \cdot 2^j} \circ B_{i \cdot 2^j + 1} \cdots B_{(i+1) \cdot 2^j - 1} = B_{2i-1,j-1} \circ B_{2i,j-1}.$$

If $A$ is a node, then $\tilde{A}$ is the program obtained by using the INW generator on $A$. Formally, we define $\tilde{B}_{i,j}$ recursively by $\tilde{B}_{i,0} = B_i$ and

$$\tilde{B}_{i,j+1} = \tilde{B}_{2i-1,j} \circ_H \tilde{B}_{2i,j},$$

where $H$ is the expander family used by the INW generator. Note that, by Lemma 2, $B_{i,j}$ and $\tilde{B}_{i,j}$ are permutation programs for all $i$ and $j$. The overall program is $B = B_{1,\log n}$ and $B$ run on the output of the INW generator is $\tilde{B}_{1,\log n}$. So our goal is to show that

$$\left|\left|\tilde{B}_{1,\log n} - B_{1,\log n}\right|\right|_{\text{Fr}} \leq \varepsilon.$$

Let $\alpha$, $\beta$, and $\gamma$ be constants (depending on $w$) to be determined later. Let $\rho(x) = \alpha\beta^{-x}$. We inductively show that for each node $B_{i,j}$ we have

$$\left|\left|\tilde{B}_{i,j} - B_{i,j}\right|\right|_{\text{Fr}} \leq \rho(||B_{i,j}||_{\text{Fr}}^2).$$

Label each tree edge $(B_{2i-1,j-1}, B_{i,j})$ or $(B_{2i,j-1}, B_{i,j})$ with the length $||B_{2i-1,j-1}||_{\text{Fr}}^2 - ||B_{i,j}||_{\text{Fr}}^2$ or $||B_{2i,j-1}||_{\text{Fr}}^2 - ||B_{i,j}||_{\text{Fr}}^2$ respectively. The edge length represents the amount of randomness added by the composition. Note that, by Lemma 6, these lengths are always non-negative.

For each node, remove the longer edge (breaking ties arbitrarily) and, if the longer edge is shorter than $\gamma^2$, remove both edges. Now only disjoint paths remain.

We will use the Key Convergence Lemma (Theorem 12 “Mixing Kills Error”) to bound the error buildup along these paths. The induction hypothesis and Proposition 10 (“Introducing Error Requires Introducing Randomness”) are used to bound the error along removed edges.

Choose a node $B_{i,j}$ and consider the path $B_{i_0,j}, B_{i_1,j-1}, B_{i_2,j-2} \cdots B_{i_l,j-l}$ (possibly of trivial length $l = 0$) starting at $B_{i_0,j} = B_{i,j}$. Let $x = ||B_{i,j}||_{\text{Fr}}$.

By Proposition 18, the error accumulated along the path is

$$\left|\left|\tilde{B}_{i,j} - B_{i,j}\right|\right|_{\text{Fr}} \leq g(w)(\lambda_H + \delta) + 2\left|\left|\tilde{B}_{i_l,j-l} - B_{i_l,j-l}\right|\right|_{\text{Fr}},$$

where

$$g(w) = 4w(2w)!\left(\frac{3}{1-\lambda_{2w}}\right)^{2w} = 2^{O(w \log w)}$$

and $\delta$ is an upper bound on the error of any node to the side of the path. Choose $i'_s$ such that, for $0 \leq s < l$, either

$$B_{i_s,j-s} = B_{i'_s,j-s-1} \circ B_{i_{s+1},j-s-1} \quad \text{or} \quad B_{i_s,j-s} = B_{i_{s+1},j-s-1} \circ B_{i',j-s-1}.$$

We need to bound the error on nodes that are off the side of the path—that is, we need to find $\delta$ such that $\left|\left|\tilde{B}_{i'_s,j-s-1} - B_{i'_s,j-s-1}\right|\right|_{\text{Fr}} \leq \delta$.

We inductively assume that each node $B_{i',j'}$ lower ($j' < j$) in the tree

$$\left|\left|\tilde{B}_{i',j'} - B_{i',j'}\right|\right|_{\text{Fr}} \leq \rho(||B_{i',j'}||^2_{\text{Fr}}).$$

Since the edge $(B_{i'_s,j-s-1}, B_{i_s,j-s})$ was cut and $(B_{i_{s+1},j-s-1}, B_{i_s,j-s})$ was not cut, the length of $(B_{i'_s,j-s-1}, B_{i_s,j-s})$ is at least $\gamma^2$. Thus

$$||B_{i'_s,j-s-1}||^2_{\text{Fr}} \geq ||B_{i_s,j-s}||^2_{\text{Fr}} + \gamma^2 \geq ||B_{i,j}||^2_{\text{Fr}} + \gamma^2 = x^2 + \gamma^2,$$

whence, by the induction hypothesis, we can take $\delta = \rho(x^2 + \gamma^2)$.

Now we bound the error at the end of the path—that is, $\left|\left|\tilde{B}_{i_l,j-l} - B_{i_l,j-l}\right|\right|_{\text{Fr}}$. If $B_{i_l,j-l}$ is a leaf in the original tree ($j - l = 0$), then $\left|\left|\tilde{B}_{i_l,j-l} - B_{i_l,j-l}\right|\right|_{\text{Fr}} = 0$. Suppose $j - l > 0$. Then $B_{i_l,j-l} = B_{2i_l-1,j-l-1} \circ B_{2i_l,j-l-1}$. Since $||B_{2i_l-1,j-l-1}||_{\text{Fr}} \leq ||B_{i,j}||_{\text{Fr}} = x$, we have $\left|\left|\tilde{B}_{2i_l-1,j-l-1} - B_{2i_l-1,j-l-1}\right|\right|_{\text{Fr}} \leq \rho(x^2)$. Similarly, $\left|\left|\tilde{B}_{2i_l,j-l-1} - B_{2i_l,j-l-1}\right|\right|_{\text{Fr}} \leq \rho(x^2)$. Thus, by Proposition 10 and Lemmas 15 and 7,

$$\begin{aligned}
\left|\left|\tilde{B}_{i_l,j-l} - B_{i_l,j-l}\right|\right|_{\text{Fr}} \leq& \left|\left|\tilde{B}_{2i_l-1,j-l-1} \circ_H \tilde{B}_{2i_l,j-l-1} - \tilde{B}_{2i_l-1,j-l-1}\tilde{B}_{2i_l,j-l-1}\right|\right|_{\text{Fr}} \\
&+ \left|\left|(\tilde{B}_{2i_l-1,j-l-1} - B_{2i_l-1,j-l-1})B_{2i_l,j-l-1}\right|\right|_{\text{Fr}} \\
&+ \left|\left|B_{2i_l-1,j-l-1}(\tilde{B}_{2i_l,j-l-1} - B_{2i_l,j-l-1})\right|\right|_{\text{Fr}} \\
&+ \left|\left|(\tilde{B}_{2i_l-1,j-l-1} - B_{2i_l-1,j-l-1})(\tilde{B}_{2i_l,j-l-1} - B_{2i_l,j-l-1})\right|\right|_{\text{Fr}} \\
\leq& \lambda_H w + \frac{\gamma}{\sqrt{1-\lambda_w^2}}\left(\left|\left|\tilde{B}_{2i_l-1,j-l-1} - B_{2i_l-1,j-l-1}\right|\right|_{\text{Fr}} + \left|\left|\tilde{B}_{2i_l,j-l-1} - B_{2i_l,j-l-1}\right|\right|_{\text{Fr}}\right) \\
&+ \left|\left|\tilde{B}_{2i_l-1,j-l-1} - B_{2i_l-1,j-l-1}\right|\right|_{\text{Fr}} \left|\left|\tilde{B}_{2i_l,j-l-1} - B_{2i_l,j-l-1}\right|\right|_{\text{Fr}} \\
\leq& \lambda_H w + \frac{2\gamma}{\sqrt{1-\lambda_w^2}}\rho(x^2) + \rho(x^2)^2.
\end{aligned}$$

So it remains to show that

$$\left|\left|\tilde{B}_{i,j} - B_{i,j}\right|\right|_{\text{Fr}} \leq g(w)\left(\lambda_H + \rho(x^2+\gamma^2)\right) + 2\left(\lambda_H w + \frac{2\gamma}{\sqrt{1-\lambda_w^2}}\rho(x^2) + \rho(x^2)^2\right) \leq \rho(x^2).$$

Let

$$\begin{aligned}
&\gamma = \frac{\sqrt{1-\lambda_w^2}}{16} = \frac{1}{O(w)},\\
&\beta = (4g(w))^{\gamma^{-2}} = 2^{O(w^3\log w)}, \quad \text{and}\\
&\alpha = \lambda_H 4(g(w)+2w)\beta^w = \lambda_H 2^{O(w^4\log w)}.
\end{aligned}$$

We assume that $\alpha \leq 1/4$ or, equivalently, $\lambda_H \leq \beta^{-w}/(16g(w)) = 2^{-O(w^4\log w)}$. Then

$$\begin{aligned}
(g(w)+2w)\lambda_H =& \frac{\alpha\beta^{-w}}{4} = \frac{1}{4}\rho(w) \leq \frac{1}{4}\rho(x^2),\\
g(w)\rho(x^2+\gamma^2) =& g(w)\beta^{-\gamma^2}\rho(x^2) = \frac{1}{4}\rho(x^2),\\
\frac{4\gamma}{\sqrt{1-\lambda_w^2}}\rho(x^2) =& \frac{1}{4}\rho(x^2),\\
\rho(x^2)^2 \leq& \frac{1}{4}\rho(x^2),
\end{aligned}$$

whence

$$\begin{aligned}
&g(w)(\lambda_H + \rho(x^2+\gamma^2)) + 2(\lambda_H w + \frac{2\gamma}{\sqrt{1-\lambda_w^2}}\rho(x^2) + \rho(x^2)^2)\\
&= (g(w)+2w)\lambda_H + g(w)\rho(x^2+\gamma^2) + \frac{4\gamma}{\sqrt{1-\lambda_w^2}}\rho(x^2) + \rho(x^2)^2,\\
&\leq 4\frac{1}{4}\rho(x^2)\\
&= \rho(x^2),
\end{aligned}$$

as required. Now the final error is

$$\left|\left|\tilde{B}_{1,\log n} - B_{1,\log n}\right|\right|_{\text{Fr}} \leq \rho\left(||B_{1,\log n}||_{\text{Fr}}^2\right) \leq \rho(1) = \lambda_H 2^{O(w^4\log w)}.$$

So setting $\lambda_H = \varepsilon \cdot 2^{-O(w^4\log w)}$ suffices to prove the theorem. □

## 4 Further Work

There are a number of interesting research directions opened by our results, including the following.

- We would like to generalise these results to regular branching programs. Much of the intuition for our result carries over to this more general setting and our linear-algebraic language allows us to reason about them. Indeed, many of our results hold for regular non-permutation branching programs. In particular, Theorem 12 (but not Proposition 18) and Proposition 10 (but not Corollary 11) carry over to this more general setting. If we could generalise Proposition 18 and Corollary 11, then we could generalise the main theorem.

- We would also like to unify our results with other results for regular and permutation branching programs, such as [2, 3, 12, 14]. There are strong intuitive connections, as we mentioned in Section 1.2. We can make some of these connections more formal. For example, we can show that, if $A$ and $B$ are regular branching programs, then

$$\sqrt{||B||_{\mathrm{Fr}}^2 - ||AB||_{\mathrm{Fr}}^2} \leq \mathrm{poly}(w) \cdot \mathrm{weight}(A),$$

  where $\mathrm{weight}(A)$ is a natural generalisation of the weight used by Braverman et al. [2].

- The two intuitions "Introducing Error Requires Introducing Randomness" and "Mixing Kills Error" are related. Unifying them may further simplify the proof and lead to a better understanding of our results.

## Acknowledgements

We wish to thank Salil Vadhan for many helpful conversations. We also wish to thank Anindya De, Omer Reingold, Rocco Servedio, and Luca Trevisan for their help.

## References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 132–140, New York, NY, USA, 1987. ACM.

[2] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:40–47, 2010.

[3] Joshua Brody and Elad Verbin. The coin problem and pseudorandomness for branching programs. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 30–39, Washington, DC, USA, 2010. IEEE Computer Society.

[4] Anindya De. Pseudorandomness for permutation and regular branching programs. In *Proceedings of the 2011 IEEE 26th Annual Conference on Computational Complexity*, CCC '11, pages 221–231, Washington, DC, USA, 2011. IEEE Computer Society.

[5] Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil Vadhan. Better pseudorandom generators from milder pseudorandom restrictions.

[6] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S.)*, 43(4):439–561 (electronic), 2006.

[7] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *In Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 356–364, 1994.

[8] Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 263–272, New York, NY, USA, 2011. ACM.

[9] Noam Nisan. $\mathcal{RL} \subset \mathcal{SC}$. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 619–623, New York, NY, USA, 1992. ACM.

[10] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *In Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 159–168, 1999.

[11] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 3 –13, 2000.

[12] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008.

[13] Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the $\mathcal{RL}$ vs. $\mathcal{L}$ problem. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 457–466, New York, NY, USA, 2006. ACM.

[14] Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Proceedings of the 8th international workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th international conference on Randamization and Computation: algorithms and techniques*, APPROX'05/RANDOM'05, pages 436–447, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] Michael Saks and Shiyu Zhou. $\mathrm{BP_HSPACE}(S) \subset \mathrm{DSPACE}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376 – 403, 1999.

[16] Salil Vadhan. Pseudorandomness. `http://people.seas.harvard.edu/~salil/pseudorandomness/`.

[17] Jiří Šíma and Stanislav Žák. A sufficient condition for sets hitting the class of read-once branching programs of width 3. In *Proceedings of the 38th international conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM'12, pages 406–418, Berlin, Heidelberg, 2012. Springer-Verlag.

The most up-to-date version of this paper will be made available at

`http://people.seas.harvard.edu/~tsteinke/frobenius/`