Pseudodistributions That Beat All Pseudorandom Generators (Extended Abstract)

Edward Pyne
Harvard University, Cambridge, MA, USA

Salil Vadhan
Harvard University, Cambridge, MA, USA

Abstract

A recent paper of Braverman, Cohen, and Garg (STOC 2018) introduced the concept of a *weighted pseudorandom generator (WPRG)*, which amounts to a pseudorandom generator (PRG) whose outputs are accompanied with real coefficients that scale the acceptance probabilities of any potential distinguisher. They gave an explicit construction of WPRGs for ordered branching programs whose seed length has a better dependence on the error parameter ε than the classic PRG construction of Nisan (STOC 1990 and Combinatorica 1992).

In this work, we give an explicit construction of WPRGs that achieve parameters that are *impossible* to achieve by a PRG. In particular, we construct a WPRG for *ordered permutation branching programs of unbounded width* with a single accept state that has seed length $\tilde{O}(\log^{3/2} n)$ for error parameter $\varepsilon = 1/\operatorname{poly}(n)$, where n is the input length. In contrast, recent work of Hoza et al. (ITCS 2021) shows that any PRG for this model requires seed length $\Omega(\log^2 n)$ to achieve error $\varepsilon = 1/\operatorname{poly}(n)$.

As a corollary, we obtain explicit WPRGs with seed length $\tilde{O}(\log^{3/2} n)$ and error $\varepsilon = 1/\operatorname{poly}(n)$ for ordered permutation branching programs of width $w = \operatorname{poly}(n)$ with an arbitrary number of accept states. Previously, seed length $o(\log^2 n)$ was only known when both the width and the reciprocal of the error are subpolynomial, i.e. $w = n^{o(1)}$ and $\varepsilon = 1/n^{o(1)}$ (Braverman, Rao, Raz, Yehudayoff, FOCS 2010 and SICOMP 2014).

The starting point for our results are the recent space-efficient algorithms for estimating random-walk probabilities in directed graphs by Ahmadenijad, Kelner, Murtagh, Peebles, Sidford, and Vadhan (FOCS 2020), which are based on spectral graph theory and space-efficient Laplacian solvers. We interpret these algorithms as giving WPRGs with large seed length, which we then derandomize to obtain our results. We also note that this approach gives a simpler proof of the original result of Braverman, Cohen, and Garg, as independently discovered by Cohen, Doron, Renard, Sberlo, and Ta-Shma (these proceedings).

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases pseudorandomness, space-bounded computation, spectral graph theory

Digital Object Identifier 10.4230/LIPIcs.CCC.2021.33

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2021/019/` [28]

Funding *Edward Pyne*: Supported by NSF grant CCF-1763299.
Salil Vadhan: Supported by NSF grant CCF-1763299 and a Simons Investigator Award.

Acknowledgements We thank Jack Murtagh and Sumegha Garg for insightful discussions, and Oded Goldreich and the CCC '21 reviewers for feedback that improved our presentation.

1 Introduction

The notion of a **pseudorandom generator (PRG)** [7, 35, 26] is ubiquitous in theoretical computer science, with vast applicability in cryptography and derandomization. (See the texts [17, 34] for more background on pseudorandomness.) A recent work of Braverman, Cohen, and Garg [9] introduced the following intriguing generalization of a PRG, in which we attach real coefficients to the outputs of the generator:

▶ **Definition 1.** *Let $\mathcal{B}$ be a class of boolean functions $B\colon \{0,1\}^n \to \{0,1\}$. An **$\varepsilon$-weighted pseudorandom generator (WPRG)** for $\mathcal{B}$ is a function $(G,\rho)\colon \{0,1\}^s \to \{0,1\}^n \times \mathbb{R}$ such that for every $B \in \mathcal{B}$,*

$$\left| \mathop{\mathbb{E}}_{x \leftarrow U_{\{0,1\}^n}}[B(x)] - \mathop{\mathbb{E}}_{x \leftarrow U_{\{0,1\}^s}}[\rho(x) \cdot B(G(x))] \right| \le \varepsilon.$$

*The value s is the **seed length** of the WPRG, and n is the **output length** of the WPRG. We say that the WPRG is **(mildly)**[1] **explicit** if given x, $G(x)$ and $\rho(x)$ are computable in space $O(s)$, and $\rho(x)$ has absolute value at most $2^{O(s)}$.*

Above and throughout, we use the standard definition of space-bounded complexity, which counts the working, read-write memory of the algorithm, and does not include the length of the read-only input or write-only output, which can be exponentially longer than the space bound.

In the original work of Braverman, Cohen, and Garg [9] and previous versions of this paper [28], generators as above were called **pseudorandom pseudodistributions (PRPDs)**. The terminology of weighted pseudorandom generators (WPRGs) was introduced by Cohen et al. [14], and we find it more intuitive (and it avoids the double use of the "pseudo-" prefix).

With Definition 1, a PRG is a special case of a WPRG with $\rho(x) = 1$. The power of WPRGs comes from allowing the coefficients to be negative, which yields cancellations. Indeed, an explicit ε-WPRG with seed length s in which all of the coefficients are nonnegative can be converted into an explicit $O(\varepsilon)$-PRG with seed length $O(s + \log(1/\varepsilon))$. A general WPRG can be converted into a linear combination of two unweighted generators. That is, for every explicit WPRG $(G,\rho)\colon \{0,1\}^s \to \{0,1\}^n \times \mathbb{R}$, there are explicit generators $G_+ : \{0,1\}^{s'} \to \{0,1\}^n$ and $G_- : \{0,1\}^{s'} \to \{0,1\}^n$ with seed length $s' = O(s + \log(1/\varepsilon))$ and coefficients $\rho_+, \rho_- \in \mathbb{R}$ such that for every function $B : \{0,1\}^n \to \{0,1\}$, we have:

$$\mathop{\mathbb{E}}_{x}[\rho(x) \cdot B(G(x))] = \rho_+ \cdot \mathop{\mathbb{E}}_{x}[G_+(x)] - \rho_- \cdot \mathop{\mathbb{E}}_{x}[G_-(x)] \pm \varepsilon.$$

The motivation for WPRGs is that they can be used to derandomize algorithms in the same way as a PRG: we can estimate the acceptance probability of any function $B \in \mathcal{B}$ by enumerating over the seeds x of the WPRG (G,ρ) and calculating the average of the values $\rho(x) \cdot B(G(x))$. Furthermore, [9] observe that if (G,ρ) is an ε-WPRG for a model then G is an **ε-hitting set generator (HSG)**. That is, if B is any function in $\mathcal{B}$ with $\Pr[B(U_n) = 1] > \varepsilon$, then there exists an $x \in \{0,1\}^s$ such that $B(G(x)) = 1$.

Given this motivation, it is natural to ask whether WPRGs are more powerful than PRGs. That is, can ε-WPRGs achieve a shorter seed length than ε-PRGs for a natural computational model $\mathcal{B}$? (There are simple constructions of artificial examples.) As discussed below, Braverman, Cohen, and Garg [9] gave an explicit construction of WPRGs achieving a shorter seed length than the *best known* construction of PRGs for ordered branching programs, but not beating the best possible seed length for that model (given by a non-explicit application of the Probabilistic Method). In this work, we give an explicit construction of WPRGs for a natural computational model (ordered permutation branching programs of unbounded width) with a seed length that beats all possible PRGs for that model.

[1] We consider this definition to correspond to *mild* explicitness because requiring that the generator be computable in space linear in its seed length only implies that it is computable in time exponential in its seed length (i.e. time polynomial in the size of its truth table), which is mildly explicit according to the terminology in [34]. *Strong* explicitness, in contrast, would require that each bit of the truth table is computable in time polynomial in s.

1.1 Ordered Branching Programs

The work of Braverman, Cohen, and Garg [9], as well as our paper, focuses on WPRGs for classes $\mathcal{B}$ of functions computable by *ordered branching programs*, a nonuniform model that captures how a space-bounded randomized algorithm accesses its random bits.

▶ **Definition 2.** *An **(ordered) branching program** B of length n and width w computes a function $B : \{0,1\}^n \to \{0,1\}$. On an input $\sigma \in \{0,1\}^n$, the branching program computes as follows. It starts at a fixed start state $v_0 \in [w]$. Then for $r = 1, \ldots, n$, it reads the next symbol σ_r and updates its state according to a transition function $B_r : [w] \times \{0,1\} \to [w]$ by taking $v_t = B_r(v_{t-1}, \sigma_t)$. Note that the transition function B_r can differ at each time step.*

*The branching program **accepts** σ, denoted $B(\sigma) = 1$, if $v_n \in V_{acc}$, where $V_{acc} \subseteq [w]$ is the set of accept states, and otherwise it **rejects**, denoted $B(\sigma) = 0$. Thus an ordered branching program is specified by the transition functions $B_1, \ldots, B_n$, the start state v_0 and the set V_{acc} of accept states.*

An ordered branching program of length n and width w can compute the output of an algorithm that uses $\log w$ bits of memory and n random bits, by taking the state at each layer as the contents of memory at that time. We note that we can convert any ordered branching program into one with a single accept state by collapsing all of V_{acc} to a single state.

Using the probabilistic method, it can be shown that there *exists* an ε-PRG for ordered branching programs of length n and width w with seed length $s = O(\log(nw/\varepsilon))$. The classic construction of Nisan [25] gives an explicit PRG with seed length $s = O(\log n \cdot \log(nw/\varepsilon))$, and this bound has not been improved except for extreme ranges of w, namely when w is at least quasipolynomially larger than (n/ε) [27, 5, 22] or when $w \leq 3$ [8, 32, 19, 24]. Braverman, Cohen, and Garg [9] gave an explicit construction of a WPRG that achieves improved dependence on the error parameter ε, with seed length

$$s = \tilde{O}\left(\log n \cdot \log(nw) + \log(1/\varepsilon)\right).$$

In particular, for error $\varepsilon = n^{-\log n}$ and width $w = \mathrm{poly}(n)$, their seed length improves Nisan's from $O(\log^3 n)$ to $\tilde{O}(\log^2 n)$. Chatthopadhyay and Liao [12] gave a simpler construction of WPRGs with a slightly shorter seed length than [9], with an additive dependence on $O(\log(1/\varepsilon))$ rather than $\tilde{O}(\log(1/\varepsilon))$.

1.2 Permutation Branching Programs

Due to the lack of progress in constructing improved PRGs for general ordered branching programs as well as some applications, attention has turned to more restricted classes of ordered branching programs. In this work, our focus is on *permutation* branching programs:

▶ **Definition 3.** *An **(ordered) permutation branching program** is an ordered branching program B where for all $t \in [n]$ and $\sigma \in \{0,1\}$, $B_t(\cdot, \sigma)$ is a permutation on $[w]$.*

This can be thought of as the computation being time-reversible on any fixed input σ. We note that we cannot assume without loss of generality that a permutation branching program has a single accept state, as merging a set of accept states will destroy the permutation property. Nevertheless, ordered permutation branching programs with a single accept state can compute interesting functions, such as testing whether a $\sum_{i \in S} x_i \equiv 0 \pmod{m}$, for any $m \leq w$ and any $S \subseteq [n]$. An ordered permutation branching program with a single accept state can also test whether $x|_T = \pi(x|_S)$ for any permutation $\pi : \{0,1\}^\ell \to \{0,1\}^\ell$ and any two subsets $S, T \subseteq [n]$ of size ℓ such that all elements of T are larger than all elements of S, provided that $w \geq 2^\ell$ [20].

Previous works on various types of PRGs for permutation branching programs [30, 29, 10, 23, 15, 33, 20] have achieved seed lengths that are logarithmic or nearly logarithmic in the length n of the branching program, improving the $\log^2 n$ bound in Nisan's generator. In particular, Braverman, Rao, Raz, and Yehudayoff [10] gave a PRG for the more general model of *regular* branching programs (with an arbitrary number of accept states) with seed length

$$s = O\left(\log n \cdot (\log w + \log(1/\varepsilon) + \log\log n)\right).$$

For getting a HSG, they also showed how how to eliminate the $\log\log n$ and $\log(1/\varepsilon)$ terms at the price of a worse dependence on w,[2] achieving a seed length of

$$s \le \log(n+1) \cdot w.$$

For the specific case of permutation branching programs, Koucký, Nimbhorkar, and Pudlák [23], De [15], and Steinke [33] showed how to remove the $\log\log n$ term in the Braverman et al. PRG at the price of a worse dependence on w, achieving seed length

$$s = O(\log n \cdot (\mathrm{poly}(w) + \log(1/\varepsilon))).$$

Most recently, Hoza, Pyne, and Vadhan [20] showed that the dependence on the width w could be entirely eliminated if we restrict to permutation branching programs with a *single accept state*, constructing a PRG with seed length

$$s = O(\log n \cdot (\log\log n + \log(1/\varepsilon)).$$

In particular, they show that this seed length is provably better than what is achieved by the Probabilistic Method; that is, a random function with seed length $o(n)$ fails to be a PRG for unbounded-width permutation branching programs with high probability. Like the prior PRGs for bounded-width permutation branching programs, the seed length has a term of $O(\log n \cdot \log(1/\varepsilon))$. However, in contrast to the bounded-width case, this cannot be improved to $O(\log(n/\varepsilon))$ by a non-explicit construction. Hoza et al. prove that seed length $\Omega(\log n \cdot \log(1/\varepsilon))$ is *necessary* for any ε-PRG against unbounded-width permutation branching programs. For hitting-set generators (HSGs), they show that seed length $O(\log(n/\varepsilon))$ is possible via the Probabilistic Method, thus leaving an explicit construction as an open problem.

1.3 Our Results

In this paper, we construct an explicit WPRG for permutation branching programs of unbounded width and a single accept state that beats the aforementioned lower bounds for PRGs:

▶ **Theorem 4.** *For all $n \in N$ and $\varepsilon \in (0, 1/2)$, there is an explicit ε-WPRG (and hence ε-HSG) for ordered permutation branching programs of length n, arbitrary width, and a single accept state, with seed length*

$$s = O\left(\log(n)\sqrt{\log(n/\varepsilon)}\sqrt{\log\log(n/\varepsilon)} + \log(1/\varepsilon)\log\log(n/\varepsilon)\right).$$

In particular, when $\varepsilon = 1/\mathrm{poly}(n)$, we achieve seed length $\tilde{O}(\log^{3/2} n)$, while a PRG requires seed length $\Omega(\log^2 n)$ [20].

[2] The lack of dependence on ε can be explained by the observation of Braverman et al. that any regular branching program that has nonzero acceptance probability has acceptance probability at least $1/2^{w-1}$, so WLOG $\varepsilon > 1/2^w$, i.e. $w > \log(1/\varepsilon)$.

As noted in [20], an ε-WPRG for branching programs with a single accept state is also an $(a \cdot \varepsilon)$-WPRG for branching programs with at most a accept states. For bounded-width permutation branching programs, we can take $a = w$ and obtain:

▶ **Corollary 5.** *For all $n, w \in \mathbb{N}$ and $\varepsilon \in (0, 1/2)$, there is an explicit ε-WPRG (and hence ε-HSG) for ordered permutation branching programs of length n and width w (and any number of accept states), with seed length*

$$s = O\left(\log(n)\sqrt{\log(nw/\varepsilon)}\sqrt{\log\log(nw/\varepsilon)} + \log(w/\varepsilon)\log\log(nw/\varepsilon)\right).$$

In particular for $w = \operatorname{poly}(n)$ and $\varepsilon = 1/\operatorname{poly}(n)$, we achieve seed length $\tilde{O}(\log^{3/2} n)$. Note that the previous explicit PRGs (or even HSGs) for permutation branching programs (as mentioned in Subsection 1.2) achieved seed length $o(\log^2 n)$ only when both $w = n^{o(1)}$ and $\varepsilon = 1/n^{o(1)}$. With seed length $o(\log^2 n)$, Corollary 5 can handle width as large as $w = n^{\tilde{\Omega}(\log n)}$ and error as small as $\varepsilon = 1/n^{-\tilde{\Omega}(\log(n))}$. We summarize these results in a table.

Citation	Type	Model	Seed Length
Non-explicit (folklore)	PRG	General	$\Theta(\log(nw/\varepsilon)$
[25, 21]	PRG	General	$O(\log n \cdot \log(nw/\varepsilon))$
[10]	PRG	Regular	$\tilde{O}(\log n \cdot \log(w/\varepsilon))$
[10]	HSG	Regular	$\log(n+1) \cdot w$
[23, 15, 33]	PRG	Permutation	$O(\log n \cdot (\operatorname{poly}(w) + \log(1/\varepsilon))$
[9, 12, 28]	WPRG	General	$\tilde{O}(\log n \cdot \log nw + \log(1/\varepsilon))$
[20]	PRG	Permutation (1 accept)	$\tilde{\Theta}(\log n \cdot \log(1/\varepsilon))$
Non-explicit [20]	HSG	Permutation (1 accept)	$O(\log(n/\varepsilon))$
Theorem 4	WPRG	Permutation (1 accept)	$\tilde{O}(\log n\sqrt{\log(n/\varepsilon)} + \log(1/\varepsilon))$
Corollary 5	WPRG	Permutation	$\tilde{O}(\log n\sqrt{\log(nw/\varepsilon)} + \log(w/\varepsilon))$

2 Overview of Proofs

The starting point for our results are the recent space-efficient algorithms for estimating random-walk probabilities in directed graphs by Ahmadenijad, Kelner, Murtagh, Peebles, Sidford, and Vadhan [2], which are based on spectral graph theory and space-efficient Laplacian solvers. We interpret these algorithms as giving WPRGs with large seed length, which we then derandomize to obtain our results.

The specific problem considered by Ahmadenijad et al. is the following: given a directed graph $\mathcal{G} = (V, E)$, two vertices $s, t \in V$, a walk-length $k \in \mathbb{N}$, and an error parameter $\varepsilon > 0$, estimate the probability that a random walk of length k started at s ends at t to within $\pm\varepsilon$. Such an algorithm can be applied to the following graph in order to estimate the acceptance probability of an ordered branching program:

▶ **Definition 6.** *Given a length n, width w branching program B with transition functions $(B_1, \ldots, B_n)$ with start vertex $v_0 \in [w]$, and a single accept vertex v_{acc}, the **(layered) graph associated with B** is the graph $\mathcal{G}$ with vertex set $\{0, 1, \ldots, n\} \times [w]$ and directed edges from $(i-1, v)$ to $(i, B_i(v, 0))$ and $(i, B_i(v, 1))$ for every $i = 1, \ldots, n$ and $v \in [w]$.*

Applying the algorithms of Ahmadenijad et al. to the graph $\mathcal{G}$ with $s = (0, v_0)$, $t = (n, v_{\text{acc}})$, and $k = n$, we obtain an estimate of the acceptance probability of B to within $\pm\varepsilon$, just like an ε-WPRG for B would allow us to obtain. But a WPRG (G, ρ) is much more constrained than an arbitrary space-efficient algorithm, which can directly inspect the graph. Instead, a WPRG is limited to generating $S = 2^s$ walks of length n in the layered graph, described by sequences $G(x_1), \ldots, G(x_S) \in \{0,1\}^n$ of edge labels, and then combining the indicators $B(G(x_1)), \ldots, B(G(x_n))$ of whether the walks ended at t via a linear combination with fixed coefficients $\rho(x_1), \ldots, \rho(x_S) \in \mathbb{R}$.

Note that if B is a permutation branching program, then the graph $\mathcal{G}$ above is 2-regular (except for layer 0 which has no incoming edges and layer n which has no outgoing edges). Thus, the basis for Theorem 4 is the (main) result of Ahmadenijad et al., which applies to regular (or more generally, Eulerian) directed graphs G. However, they also give a new algorithm for estimating random-walk probabilities in arbitrary directed graphs. This algorithm is not as space-efficient as the ones for regular graphs, but is significantly simpler, so we begin by describing how to obtain a WPRG based on that algorithm. The resulting WPRG matches the parameters of the WPRG of Braverman, Cohen, and Garg [9], but has a significantly simpler proof (and is also simpler than the construction of Chatthopadhyay and Liao [12]). A similar construction was independently discovered by Cohen, Doron, Renard, Sberlo, and Ta-Shma [14].

2.1 WPRG for Arbitrary Ordered Branching Programs

Let B be an arbitrary width w, length n ordered branching program, with associated layered graph $\mathcal{G}$ as in Definition 6. The algorithm of Ahmadenijad et al. starts with the $(n+1)w \times (n+1)w$ random-walk transition matrix $\mathbf{W}$ of $\mathcal{G}$, which has the following block structure:

$$\mathbf{W} = \begin{bmatrix} 0 & \mathbf{B}_1 & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{B}_2 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \mathbf{B}_n \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Here entry $((i,u),(j,v))$ is the probability that taking one random step in $\mathcal{G}$ from vertex (i,u) ends at (j,v). Thus $\mathbf{B}_i$ is the $w \times w$ transition matrix for the random walk from layer $i-1$ to i in the branching program. (Note that the matrix $\mathbf{W}$ is not quite stochastic due to layer n having no outgoing edges.)

Ahmadenijad et al. consider the Laplacian $\mathbf{L} = \mathbf{I}_{(n+1)w} - \mathbf{W}$. Its inverse $\mathbf{L}^{-1} = (\mathbf{I}_{(n+1)w} - \mathbf{W})^{-1} = \mathbf{I}_{(n+1)w} + \mathbf{W} + \mathbf{W}^2 + \mathbf{W}^3 + \cdots$ sums up random-walks of all lengths in G, and thus has the following form:

$$\mathbf{L}^{-1} = \begin{bmatrix} \mathbf{B}_{0\ldots0} & \mathbf{B}_{0\ldots1} & \mathbf{B}_{0\ldots2} & \cdots & \mathbf{B}_{0\ldots n} \\ 0 & \mathbf{B}_{1\ldots1} & \mathbf{B}_{1\ldots2} & \cdots & \mathbf{B}_{1\ldots n} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \mathbf{B}_{n-1\ldots n} \\ 0 & 0 & 0 & \cdots & \mathbf{B}_{n\ldots n} \end{bmatrix},$$

where

$$\mathbf{B}_{i\ldots j} = \mathbf{B}_{i+1}\mathbf{B}_{i+2}\cdots\mathbf{B}_j.$$

In particular, the $(0,n)$'th block of $\mathbf{L}^{-1}$ gives the random-walk probabilities from layer 0 to layer n, and thus the acceptance probability of G is exactly the (v_0, v_{acc})'th entry of the $(0,n)$'th block of $\mathbf{L}^{-1}$. Therefore, the task reduces to producing a sufficiently good estimate of $\mathbf{L}^{-1}$.

Ahmadenijad et al. estimate $\mathbf{L}^{-1}$ in two steps. First, they observe that the Saks–Zhou derandomization of logspace [31] can be used to produce, in deterministic space $O(\log(nw)\sqrt{\log(n)})$, approximations $\widetilde{\mathbf{B}_{i\dots j}}$ of the blocks $\mathbf{B}_{i\dots j}$ to within entrywise error $1/\operatorname{poly}(nw)$, resulting in an approximate pseudoinverse

$$\widetilde{\mathbf{L}^{-1}} = \begin{bmatrix} \widetilde{\mathbf{B}_{0\dots0}} & \widetilde{\mathbf{B}_{0\dots1}} & \widetilde{\mathbf{B}_{0\dots2}} & \cdots & \widetilde{\mathbf{B}_{0\dots n}} \\ 0 & \widetilde{\mathbf{B}_{1\dots1}} & \widetilde{\mathbf{B}_{1\dots2}} & \cdots & \widetilde{\mathbf{B}_{1\dots n}} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \widetilde{\mathbf{B}_{n-1\dots n}} \\ 0 & 0 & 0 & \cdots & \widetilde{\mathbf{B}_{n\dots n}} \end{bmatrix}, \tag{1}$$

with the property that

$$\left\|\mathbf{I}_{(n+1)w} - \widetilde{\mathbf{L}^{-1}}\mathbf{L}\right\|_1 \le 1/nw,$$

where $\|\cdot\|_1$ denotes the ℓ_1 operator norm on row vectors, ie $\|\mathbf{M}\|_1 = \sup_{x\neq0}\|x\mathbf{M}\|_1/\|x\|_1$.

Next, Ahmadenijad et al. reduce the approximation error to an arbitrary $\varepsilon < 1/(nw)^{O(1)}$ by using preconditioned Richardson iterations, as captured by the following lemma:

▶ **Lemma 7** (preconditioned Richardson iteration, [2] Lemma 6.2). *Let $\|\cdot\|$ be a submultiplicative norm on $N\times N$ real matrices. Given matrices $\mathbf{A}, \mathbf{P}_0 \in \mathbb{R}^{N\times N}$ such that $\|\mathbf{I}_N - \mathbf{P}_0\mathbf{A}\| \le \alpha$ for some constant $\alpha > 0$, let $\mathbf{P}_m = \sum_{i=0}^{m}(\mathbf{I}_N - \mathbf{P}_0\mathbf{A})^i\mathbf{P}_0$. Then $\|\mathbf{I}_N - \mathbf{P}_m\mathbf{A}\| \le \alpha^{m+1}$.*

Setting $N = (n+1)w$, $\mathbf{A} = \mathbf{L}$, $\mathbf{P}_0 = \widetilde{\mathbf{L}^{-1}}$, and $\alpha = 1/nw$, and $m = O(\log_{nw}(1/\varepsilon))$, we obtain $\widetilde{\mathbf{L}_\varepsilon} = \mathbf{P}_m$ such that $\|\mathbf{I}_N - \widetilde{\mathbf{L}_\varepsilon}\mathbf{L}\|_1 \le \varepsilon/(nw)^{O(1)}$, which implies that $\widetilde{\mathbf{L}_\varepsilon}$ and $\mathbf{L}^{-1}$ are entrywise equal up to $\pm\varepsilon$, for

$$\widetilde{\mathbf{L}_\varepsilon} = \sum_{i=0}^{m}(\mathbf{I}_N - \widetilde{\mathbf{L}^{-1}}\mathbf{L})^i\widetilde{\mathbf{L}^{-1}} \tag{2}$$

In particular, the (v_0, v_{acc})'th entry of the $(0,n)$'th block of $\widetilde{\mathbf{L}_\varepsilon}$ is an estimate of the acceptance probability of the branching program to within $\pm\varepsilon$. Computing $\widetilde{\mathbf{L}_\varepsilon}$ from $\mathbf{L}$ and $\widetilde{\mathbf{L}^{-1}}$ can be done in space $O((\log nw)\cdot\log m)$, yielding Ahmadenijad et al.'s space bound of

$$O(\log(nw)\sqrt{\log(n)} + (\log nw)\cdot\log\log_{nw}(1/\varepsilon)).$$

Now we show how, with appropriate an modification, we can interpret this algorithm of Ahmadenijad et al. as a WPRG (albeit with large seed length). We replace the use of the Saks–Zhou algorithm (which requires looking at the branching program) with Nisan's pseudorandom generator. Specifically, we take $\widetilde{\mathbf{B}_{i\dots j}}$ to be the matrix whose (u,v)'th entry is the probability that, if we start at state u in the the i'th layer and use a random output of Nisan's pseudorandom generator to take $j-i$ steps in the branching program, we end at state v in the j'th layer. For $\widetilde{\mathbf{B}_{i\dots j}}$ to approximate $\mathbf{B}_{i\dots j}$ to within error $\pm1/\operatorname{poly}(nw)$ as above, Nisan's pseudorandom generator requires seed length

$$s_{\text{Nisan}} = O(\log(j-i)\cdot\log nw) = O(\log n\cdot\log nw).$$

Observe that for every i, $\widetilde{\mathbf{B}_{i\dots i}} = \mathbf{I}_w = \mathbf{B}_{i\dots i}$. Without loss of generality, we may also assume that $\widetilde{\mathbf{B}_{(i-1)\dots i}} = \mathbf{B}_{(i-1)\dots i}$, since taking one step only requires one random bit.

Next, we observe from Equation 2 that the matrix $\widetilde{\mathbf{L}_\varepsilon}$ is a polynomial of degree $2m+1$ in the matrices $\mathbf{L}$ and $\widetilde{\mathbf{L}^{-1}}$. In particular the $(0,n)$'th block of $\widetilde{\mathbf{L}_\varepsilon}$ is a polynomial of degree at most $2m+1$ in the matrices $\widetilde{\mathbf{B}_{i\ldots j}}$. Specifically, using the upper-triangular structure of the matrices $\mathbf{L}$ and $\widetilde{\mathbf{L}^{-1}}$ and noting that the product of d $(n+1)\times(n+1)$ block matrices expands into a sum of $(n+1)^{d-1}$ terms, each of which is a product of d individual blocks, we show:

▶ **Observation 8.** *The $(0,n)$'th block of $\widetilde{\mathbf{L}_\varepsilon}$ is equals the sum of at most $(n+1)^{O(m)}$ terms, each of which is of the form*

$$\pm\widetilde{\mathbf{B}_{i_0\cdots i_1}}\widetilde{\mathbf{B}_{i_1\cdots i_2}}\cdots\widetilde{\mathbf{B}_{i_{r-1}\cdots i_r}}, \tag{3}$$

where $0=i_0<i_1<i_2<\cdots<i_r=n$ and $r\le 2m+1$.

Notice that, up to the sign, each term as expressed in Equation (3) is the transition matrix for a pseudorandom walk from layer 0 to layer n of the branching program, where we use $r\le m+1$ independent draws from Nisan's generator, with the j'th draw being used to walk from layer i_{j-1} to layer i_j. In particular, the (v_0,v_{acc}) entry of Equation (3) equals the acceptance probability of the branching program on such a pseudorandom walk (up to the $\pm$ sign). Thus the algorithm now has the form required of a WPRG.

The seed length for the WPRG is the sum of the seed length s_{sum} needed to select a random term in the sum (using the coefficients of the WPRG to rescale the sum into a expectation) and the seed length s_{term} to generate a walk for the individual term. To select a random term in the sum requires a seed of length

$$s_{\text{sum}}=\log n^{O(m)}=O(m\cdot\log(n))=O(\log_{nw}(1/\varepsilon)\cdot\log(n))=O(\log(1/\varepsilon)).$$

The seed length needed for an individual term is at most

$$s_{\text{term}}=O(m)\cdot s_{\text{Nisan}}=O(\log_{nw}(1/\varepsilon)\cdot\log(n)\cdot\log nw)=O(\log(1/\varepsilon)\cdot\log(n)).$$

The latter offers no improvement over Nisan's PRG. (Recall that $\varepsilon<1/nw$.) To obtain a shorter seed length, we just need to derandomize the product in Equation (3). Instead of using r independent seeds, we use dependent seeds generated using the Impagliazzo–Nisan–Wigderson pseudorandom generator [21]. Specifically, we can produce a pseudorandom walk that approximates the product to within entrywise error $\pm\gamma$ using a seed of length

$$s'_{\text{term}}=s_{\text{Nisan}}+O((\log r)\cdot\log(rw/\gamma)).$$

The entrywise error of γ in each term may accumulate over the $n^{O(m)}$ terms, so to achieve a WPRG error of $O(\varepsilon)$, we should set $\gamma=\varepsilon/n^{O(m)}=1/\varepsilon^{O(1)}$. Recalling that $r\le 2m+1=O(\log_{nw}(1/\varepsilon))$, we attain a seed length of

$$\begin{aligned}s_{\text{sum}}+s'_{\text{term}} &= O(\log(1/\varepsilon))+O(\log n\cdot\log nw)+O(\log\log_{nw}(1/\varepsilon)\cdot\log(1/\varepsilon))\\ &= O(\log n\cdot\log nw+\log(1/\varepsilon)\cdot\log\log_{nw}(1/\varepsilon)),\end{aligned}$$

which slightly improves over the bound of Braverman, Cohen, and Garg [9], and is incomparable to that of Chattopadhyay and Liao [12]. Specifically, our first term of $O(\log n\cdot\log nw)$ is better than [12] by a factor of $\log\log(nw)$, but our second term of $O(\log(1/\varepsilon)\cdot\log\log_{nw}(1/\varepsilon))$ is worse by a factor of $\log\log_{nw}(1/\varepsilon)$.

2.2 WPRG for Permutation Branching Programs

Now we give an overview of our WPRG for permutation branching programs, as stated in Theorem 4. This is based on the the algorithm of Ahmademnijad et al. that estimates random-walk probabilities in *regular* (or even Eulerian) digraphs with better space complexity than the algorithm described in Subsection 2.1. As before, we will review their algorithm as applied to the $((n+1)\cdot w)$-vertex graph $\mathcal{G}$ associated with an ordered branching program B of length n and width w. Since we assume that the branching program B is a permutation program, the graph $\mathcal{G}$ will be 2-regular at all layers other than 0 and n. For the spectral graph-theoretic machinery used by Ahmadenijad et al., it is helpful to work with random-walk matrices that correspond to strongly connected digraphs, so we also add a complete bipartite graph of edges from layer n back to layer 0, resulting in the following modified version of the matrix $\mathbf{W}$:

$$\mathbf{W}_0 = \begin{bmatrix} 0 & \mathbf{B}_1 & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{B}_2 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \mathbf{B}_n \\ \mathbf{J}_w & 0 & 0 & \cdots & 0 \end{bmatrix}, \tag{4}$$

where the $\mathbf{J}_w$ in the lower-left corner is the $w \times w$ matrix in which every entry is $1/w$ (corresponding to the complete bipartite graph we added). Notice that the matrix $\mathbf{J}_w$ is identically zero when applied to any vector that is orthogonal to the uniform distribution, so it is not very different than having 0 in the lower-left block as we had before. Indeed, the powers of $\mathbf{W}$ look as follows:

$$\mathbf{W}_0^2 = \begin{bmatrix} 0 & 0 & \mathbf{B}_{0..2} & 0 & 0 \\ \vdots & 0 & 0 & \ddots & 0 \\ 0 & & \vdots & & \mathbf{B}_{n-2..n} \\ \mathbf{J}_w & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{J}_w & 0 & \cdots & 0 \end{bmatrix}, \ldots, \mathbf{W}_0^n = \begin{bmatrix} 0 & 0 & \cdots & 0 & \mathbf{B}_{0..n} \\ \mathbf{J}_w & 0 & & & 0 \\ 0 & \ddots & & & 0 \\ \vdots & 0 & \mathbf{J}_w & 0 & 0 \\ 0 & 0 & 0 & \mathbf{J}_w & 0 \end{bmatrix} \tag{5}$$

where

$$\mathbf{B}_{i\ldots j} = \mathbf{B}_{i+1}\mathbf{B}_{i+2}\cdots\mathbf{B}_j.$$

Notice in particular that $\mathbf{W}_0^{n+1}$ will be a block-diagonal matrix with $\mathbf{J}_w$'s on the diagonal (i.e. $\mathbf{W}_0^{n+1} = \mathbf{I}_{n+1} \otimes \mathbf{J}_w$), and thus has no dependence on the branching program B.

Now the Laplacian $\mathbf{I}_{(n+1)w} - \mathbf{W}_0$ is no longer invertible (the uniform distribution is in the kernel). In [2], they instead estimate the Moore-Penrose pseudoinverse of $\mathbf{I}_{(n+1)w} - \mathbf{W}_0$. We instead scale $\mathbf{W}_0$ by a factor $c = 1 - 1/(n+1)$, and consider the Laplacian $\mathbf{L}_0 = \mathbf{I}_{(n+1)w} - c\mathbf{W}_0$. Looking ahead, this scaling factor ensures that the condition number of $\mathbf{L}_0$ depends only on n, allowing us to obtain a seed length independent of w. Then, by the expressions above for the powers of $\mathbf{W}_0$, it can be shown that from

$$\mathbf{L}_0^{-1} = \mathbf{I}_{(n+1)w} + c\mathbf{W}_0 + c^2\mathbf{W}_0^2 + c^3\mathbf{W}_0^3 + \ldots$$

we can compute $\mathbf{B}_{0..n}$, which appears in $\mathbf{W}_0^n$ with a scaling factor $c^n \geq 1/4$.

So again to estimate the acceptance probability of B, it suffices to compute a sufficiently good approximation to $\mathbf{L}_0^{-1}$. As before, it suffices to compute a matrix $\widetilde{\mathbf{L}_0^{-1}}$ such that $\|\mathbf{I}_N - \widetilde{\mathbf{L}_0^{-1}}\mathbf{L}_0\| \leq \alpha$ for some constant $\alpha < 1$ and a submultiplicative matrix norm $\|\cdot\|$, because then we can use preconditioned Richardson iterations (Lemma 7) to estimate $\mathbf{L}_0$ to within arbitrary entrywise accuracy.

Unfortunately, we don't know how to directly obtain such an initial approximation $\widetilde{\mathbf{L}_0^{-1}}$ efficiently enough for our result. Instead, following Ahmadenijad et al., we tensor $\mathbf{W}_0$ with a sufficiently long directed cycle. Specifically, we let $\mathbf{C}_i$ be the directed cycle on 2^i vertices, and consider $\mathbf{C}_q$ for $q = \log(n+1)$ (which we assume is an integer WLOG). We consider the *cycle lift*, whose transition matrix is

$$\mathbf{C}_q \otimes \mathbf{W}_0 = \begin{bmatrix} 0 & \mathbf{W}_0 & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{W}_0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \mathbf{W}_0 \\ \mathbf{W}_0 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

Then, we seek to invert the Laplacian $\mathbf{L} = \mathbf{I}_{2^qN} - c\mathbf{C}_q \otimes \mathbf{W}_0$. Similarly to the above, we have:

$$\begin{aligned}\mathbf{L}^{-1} &= (\mathbf{I}_{2^qN} - c\mathbf{C}_q \otimes \mathbf{W}_0)^{-1} \\ &= \left(\mathbf{I}_{2^qN} - c^{n+1}\mathbf{C}_q^{n+1} \otimes \mathbf{W}_0^{n+1}\right)^{-1} \cdot \left(\mathbf{I}_{2^qN} + c\mathbf{C}_q \otimes \mathbf{W}_0 + c^2\mathbf{C}_q^2 \otimes \mathbf{W}_0^2 + \cdots c^n\mathbf{C}_q^n \otimes \mathbf{W}_0^n\right) \\ &= \left(\mathbf{I}_{2^qN} - c^{n+1}\mathbf{C}_q^{n+1} \otimes (\mathbf{I}_{n+1} \otimes \mathbf{J}_w)\right)^{-1} \cdot \left(\mathbf{I}_{2^qN} + c\mathbf{C}_q \otimes \mathbf{W}_0 + c^2\mathbf{C}_q^2 \otimes \mathbf{W}_0^2 + \cdots c^n\mathbf{C}_q^n \otimes \mathbf{W}_0^n\right).\end{aligned}$$

Thus, letting

$$\mathbf{M} = \mathbf{I}_{2^qN} - c^{n+1}\mathbf{C}_q^{n+1} \otimes (\mathbf{I}_{n+1} \otimes \mathbf{J}_w) = \mathbf{I}_{2^qN} - c^{n+1}\mathbf{I}_{2^q} \otimes (\mathbf{I}_{n+1} \otimes \mathbf{J}_w),$$

which has no dependence on the branching program, we have:

$$\begin{aligned}\mathbf{M} \cdot \mathbf{L}^{-1} &= \mathbf{I}_{2^qN} + c\mathbf{C}_q \otimes \mathbf{W}_0 + c^2\mathbf{C}_q^2 \otimes \mathbf{W}_0^2 + \cdots c^n\mathbf{C}_q^n \otimes \mathbf{W}_0^n \\ &= \begin{bmatrix} \mathbf{I}_N & c\mathbf{W}_0 & c^2\mathbf{W}_0^2 & \cdots & c^n\mathbf{W}_0^n \\ c^n\mathbf{W}_0^n & \mathbf{I}_N & c\mathbf{W}_0 & \cdots & c^{n-1}\mathbf{W}_0^{n-1} \\ \vdots & & \ddots & & \vdots \\ c^2\mathbf{W}_0^2 & c^3\mathbf{W}_0^3 & c^4\mathbf{W}_0^4 & \ddots & c\mathbf{W}_0 \\ c\mathbf{W}_0^1 & c^2\mathbf{W}_0^2 & c^3\mathbf{W}_0^3 & \cdots & \mathbf{I}_N \end{bmatrix}\end{aligned}$$

Thus, if we can accurately estimate $\mathbf{L}^{-1}$, we can obtain an accurate estimate of $\mathbf{W}_0^n$, whose upper-right block equals $\mathbf{B}_{0..n}$ and thus contains the acceptance probability of the branching program.

To compute an approximate inverse of $\mathbf{L} = \mathbf{I}_{2^qN} - c\mathbf{C}_q \otimes \mathbf{W}_0$, Ahmadenijad et al. provide a recursive formula expressing $(\mathbf{I}_{2^qN} - c\mathbf{C}_q \otimes \mathbf{W}_0)^{-1}$ in terms of $(\mathbf{I}_{2^{q-1}N} - c^2\mathbf{C}_{q-1} \otimes \mathbf{W}_0^2)^{-1}$ and some applications of the matrix $\mathbf{W}_0$. That is, computing the inverse of the Laplacian of the cycle lift of $\mathbf{W}_0$ reduces to computing the inverse of the Laplacian of a cycle lift of $\mathbf{W}_0^2$ with a cycle of half the length. At the bottom of the recursion (after q levels of recursion), we need to compute the inverse of

$$\mathbf{I}_N - c^{2^q}\mathbf{W}_0^{2^q} = \mathbf{I}_N - c^{n+1}\mathbf{W}_0^{n+1} = \mathbf{I}_N - c^{n+1}\mathbf{I}_{n+1} \otimes \mathbf{J}_w,$$

which is easy (and does not depend on the branching program). The resulting formula for $(\mathbf{I}_{2^q N} - c\mathbf{C}_q \otimes \mathbf{W}_0)^{-1}$ is a polynomial in $\mathbf{W}_0, \mathbf{W}_0^2, \mathbf{W}_0^4, \ldots, \mathbf{W}_0^{2^{q-1}}$. However, computing these high powers of $\mathbf{W}_0$ exactly is too expensive in space usage.

Thus, instead Ahmadenijad et al. use the *derandomized square* [30] which allows for computing a sequence $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_q$ where $\mathbf{W}_i$ a sparsification of $\mathbf{W}_{i-1}^2$ with the property that $\mathbf{W}_q$ can be constructed in deterministic space

$$O(\log nw + q \cdot \log(1/\delta))$$

for an error parameter δ, rather than the space $O(q \cdot \log nw)$ of exact repeated squaring. They also introduce a new notion of spectral approximation, called *unit-circle approximation*, and show that the derandomized square $\mathbf{W}_i$ is a unit-circle approximation of $\mathbf{W}_{i-1}^2$ to within error δ. Using repeated derandomized squaring in the recursion, Ahmadenijad et al. obtain an approximate inverse $\widetilde{\mathbf{L}^{-1}}$ with the properties that:

1. The $N \times N$ blocks of $\mathbf{M} \cdot \widetilde{\mathbf{L}^{-1}}$ are each of the form $\mathbf{W}_{i_1}\mathbf{W}_{i_2} \cdots \mathbf{W}_{i_r}$ where $r = O(q)$
2. There is a submultiplicative matrix norm $\| \cdot \|_{\mathbf{F}}$ such that $\|\mathbf{I}_{2^q N} - \widetilde{\mathbf{L}^{-1}}\mathbf{L}\|_{\mathbf{F}} = O(q^2\delta)$. Moreover, achieving an $\varepsilon/\operatorname{poly}(n)$ approximation in $\mathbf{F}$-norm implies an ε approximation of $\mathbf{M} \cdot \mathbf{L}^{-1}$ in max-norm. Ahmadenijad et al. actually lose a factor of $\operatorname{poly}(nw)$ in moving from $\mathbf{F}$-norm to approximation in max-norm, but we improve this bound to $\operatorname{poly}(n)$ by our choice of scaling factor $c = 1 - 1/(n+1)$.

Item 1 allows for constructing $\mathbf{M} \cdot \widetilde{\mathbf{L}^{-1}}$ from $\mathbf{W}_0, \mathbf{W}_1, \ldots, \mathbf{W}_q$ in space

$$O(\log q \cdot \log nw).$$

By Item 2, if we take $\delta < 1/O(q^2)$, we can apply preconditioned Richardson iterations (Lemma 7) with degree $m = O(\log(n/\varepsilon)/\log(1/q\delta))$ to obtain $\widetilde{\mathbf{L}_\varepsilon} = \mathbf{P}_m$ such that $\mathbf{M} \cdot \widetilde{\mathbf{L}_\varepsilon}$ approximates $\mathbf{ML}^{-1}$ to within entrywise error ε. The preconditioned Richardson iterations have an additive space cost of:

$$O(\log m \cdot \log nw).$$

Taking $\delta = 1/O(q^2)$ and recalling that $q = \log(n+1)$, the final space complexity is

$$O(\log(nw) + q\log q) + O(\log q \cdot \log nw) + O(\log\log(n/\varepsilon) \cdot \log nw) = O(\log nw \cdot \log\log(n/\varepsilon)).$$

To view this algorithm as a WPRG for permutation branching programs, we use the equivalence between the Impagliazzo–Nisan–Wigderson (INW) generator on permutation branching programs and the derandomized square of the corresponding graph, as established in [30, 20]. Using this correspondence, the matrix $\mathbf{W}_i$ has the same structure as $\mathbf{W}^{2^i}$ (see Equation 5), except that each block of the form $\mathbf{B}_{j..j+2^i}$ is replaced with a matrix $\widetilde{\mathbf{B}_{j..j+2^i}}$ that is the transition matrix of a pseudorandom walk from layer j of the branching program to layer $j + 2^i$ using the INW generator. The seed length to generate this pseudorandom walk is

$$s_{\text{INW}} = O(q\log(q/\delta)),$$

which, as highlighted in [20], is independent of the width w of the branching program. This is the place where we use the fact that B is a permutation branching program rather than a regular branching program. Even though the algorithm Ahmadenijad et al. works for regular directed graphs (and hence regular branching programs), the derandomized square operations used in that case can no longer be viewed as being obtained by using a pseudorandom generator to derandomize walks in the graph.

Then, again assuming without loss of generality that $\widetilde{\mathbf{B}_{(j-1)\ldots j}} = \mathbf{B}_{(j-1)\ldots j}$ for $j = 1, \ldots, n$, we have the following analogue of Observation 8:

▶ **Observation 9.** *The upper-right $w \times w$ block of $\mathbf{M} \cdot \widetilde{\mathbf{L}_\varepsilon}$ equals the sum of at most $n^{O(m)}$ terms, each of which is of the form*

$$\pm\widetilde{\mathbf{B}_{i_0\cdots i_1}}\widetilde{\mathbf{B}_{i_1\cdots i_2}}\cdots\widetilde{\mathbf{B}_{i_{r-1}\cdots i_r}}, \tag{6}$$

where $0 = i_0 < i_1 < i_2 < \cdots < i_r = n$ and $r = O(qm)$.

As in Subsection 2.1, the algorithm now has the form required of a WPRG and our only remaining challenge is to keep the seed length small. The seed length for the WPRG is the sum of the seed length needed to select a random term in the sum (using the coefficients of the WPRG to rescale the sum into a expectation) and the seed length to generate a walk for the individual term. To select a random term in the sum requires a seed of length

$$s_{\text{sum}} = \log(n^{O(m)}).$$

The seed length needed for an individual term is at most

$$s_{\text{term}} = O(qm) \cdot s_{\text{INW}},$$

which again would be too expensive for us. To derandomize the product in Equation (6), we again use the INW generator, but rely on the analysis in [20] for permutation branching programs to maintain a seed length that is independent of the width. Specifically, we can produce a pseudorandom walk that approximates the product to within entrywise error $\pm\gamma$ using a seed of length

$$s'_{\text{term}} = s_{\text{INW}} + O((\log r) \cdot \log(\log(r)/\gamma)) = s_{\text{INW}} + O(\log qm \cdot \log(\log(qm)/\gamma)).$$

The entrywise error of γ in each term may accumulate over the $n^{O(m)}$ terms, so to achieve a WPRG error of $O(\varepsilon)$, we should set $\gamma = \varepsilon/n^{O(m)}$, which means that $s'_{\text{term}} \geq s_{\text{sum}}$.

All in all, we attain a seed length of

$$\begin{aligned} s_{\text{sum}} + s'_{\text{term}} &= O(m\log n) + s_{\text{INW}} + O((\log qm) \cdot \log(\log(qm)/\gamma)) \\ &= O(q\log(q/\delta)) + \tilde{O}(m\log n) + O(\log qm \cdot \log(n/\varepsilon)) \\ &= \tilde{O}\left(\log n \cdot \log(1/\delta) + \frac{\log(n/\varepsilon)}{\log(1/(\delta\log n))} \cdot \log n + \log\log(n/\varepsilon) \cdot \log(n/\varepsilon)\right) \end{aligned}$$

Optimizing the choice of δ as $\delta = \exp(-\tilde{\Theta}(\sqrt{\log(n/\varepsilon)}))$, we get a seed length of

$$\tilde{O}(\log n\sqrt{\log(n/\varepsilon)} + \log(1/\varepsilon)).$$

Note that the choice of δ here is much smaller than in the Ahmadenijad et al. algorithm, which used $\delta = 1/\operatorname{polylog}(n)$. The reason we need the smaller choice of δ is to reduce the effect of the $\log(n^{O(m)})$ price we pay in s_{sum} and s'_{term}, which does not have an analogue in the algorithm of Ahmadenijad et al.

2.3 Perspective

Some intuition for the ability of WPRGs to beat the parameters of PRGs can come from the study of *samplers* [16]. A *sampler* for a class $\mathcal{F}$ of functions $f : \{0,1\}^m \to \mathbb{R}$ is randomized algorithm Samp that is given oracle access to a function $f \in \mathcal{F}$ and, with probability at least $1-\delta$, outputs an estimate of $\mathbb{E}[f(U_n)]$ to within additive error $\pm\varepsilon$. Most often, the class $\mathcal{F}$ is taken to be the class of all bounded functions $f : \{0,1\}^m \to [0,1]$, but some works have considered the general definition and other classes, such as the class $\mathcal{F}$ of unbounded functions f such that the random variable $f(U_n)$ has subgaussian tails [6, 1]. Two key complexity parameters of a sampler are its *randomness complexity* (the number of coin tosses it uses, typically as a function of m, δ, and ε) and its *sample complexity* (the number of queries it makes to oracle f). An *averaging sampler* is one that has a restricted form, where it uses its coin tosses to generate (possibly correlated) samples $x_1, \ldots, x_S$, and then outputs the average of f on the samples, i.e. $(f(x_1) + \cdots + f(x_S))/S$.

As noted by Cheng and Hoza [13], PRGs and WPRGs can be viewed as deterministic averaging samplers (i.e. with randomness complexity and failure probability zero). Specifically, a PRG $G : \{0,1\}^s \to \{0,1\}^m$ for a class $\mathcal{F}$ is a deterministic averaging sampler for the class $\mathcal{F}$ with sample complexity $S = 2^s$. Indeed, the sampler simply outputs the set of all $S = 2^s$ outputs of G. A WPRG as a more general form of a nonadaptive deterministic sampler for the class $\mathcal{F}$, one that is restricted to output a linear combination of the function values.

So comparing the power of PRGs vs. WPRGs is a special case of the more general problem of comparing the power of averaging samplers vs. more general nonadaptive samplers. In this more general framing, there are some natural examples of classes $\mathcal{F}$ where nonadaptive samplers can have smaller sample complexity than any averaging sampler. Specifically, if we consider the class $\mathcal{F}$ of *unbounded* functions $f : \{0,1\}^m \to \mathbb{R}$ with bounded variance, i.e. $\mathrm{Var}[f(U_n)] \leq 1$, then the best sample complexity for an averaging sampler is $\Theta(\min\{1/\varepsilon^2\delta, 2^m\})$. (Essentially, Chebychev's Inequality is tight for such functions.) However, there is a nonadaptive sampler with sample complexity $O(\log(1/\delta)/\varepsilon^2)$, namely the *median-of-averages sampler*, which outputs the median of $O(\log(1/\delta))$ averages, with each average being on $O(1/\varepsilon^2)$ samples.

This example suggests two areas of investigation. First, can we gain further benefits in seed length by considering further generalizations of PRGs that are allowed to estimate acceptance probability with more general functions than linear combinations (or possibly even with adaptive queries)? Some examples are the line of work on converting hitting-set generators for circuits [3, 4, 11, 18] or ordered branching programs [13] into deterministic samplers. Second, is there a benefit in the study of samplers in restricting attention to ones that output linear combinations like WPRGs? Perhaps these still retains some of the useful composition properties and connections to other pseudorandom objects that are enjoyed by averaging samplers (cf. [36, 34, 1]), while allowing for gains in sample and/or randomness complexity.

References

1 Rohit Agrawal. Samplers and extractors for unbounded functions. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 59:1–59:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.59.

2 AmirMahdi Ahmadinejad, Jonathan A. Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil P. Vadhan. High-precision estimation of random walks in small space. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1295–1306. IEEE, 2020. doi:10.1109/FOCS46700.2020.00123.

3 Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, 1998. doi:10.1145/273865.273933.

4 Alexander E. Andreev, Andrea E. F. Clementi, José D. P. Rolim, and Luca Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116 (electronic), 1999.

5 Roy Armoni. On the derandomization of space-bounded computations. In *Randomization and approximation techniques in computer science (Barcelona, 1998)*, volume 1518 of *Lecture Notes in Comput. Sci.*, pages 47–59. Springer, Berlin, 1998.

6 Jaroslaw Blasiok. Optimal streaming and tracking distinct elements with high probability. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2432–2448. SIAM, 2018. `doi:10.1137/1.9781611975031.156`.

7 Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984. `doi:10.1137/0213053`.

8 Andrej Bogdanov, Zeev Dvir, Elad Verbin, and Amir Yehudayoff. Pseudorandomness for width 2 branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:70, 2009. URL: `http://eccc.hpi-web.de/report/2009/070`.

9 Mark Braverman, Gil Cohen, and Sumegha Garg. Hitting sets with near-optimal error for read-once branching programs. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 353–362. ACM, 2018. `doi:10.1145/3188745.3188780`.

10 Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. In *FOCS*, pages 40–47. IEEE Computer Society, 2010. `doi:10.1109/FOCS.2010.11`.

11 Harry Buhrman and Lance Fortnow. One-sided two-sided error in probabilistic computation. In *STACS 99 (Trier)*, volume 1563 of *Lecture Notes in Comput. Sci.*, pages 100–109. Springer, Berlin, 1999.

12 Eshan Chattopadhyay and Jyun-Jie Liao. Optimal error pseudodistributions for read-once branching programs. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 25:1–25:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CCC.2020.25`.

13 Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 10:1–10:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CCC.2020.10`.

14 Gil Cohen, Dean Doron, Oren Renard, Ori Sberlo, and Amnon Ta-Shma. Error reduction for weighted prgs against read once branching programs. In *36th Computational Complexity Conference, CCC 2021, July 19-23, 2021, Toronto, Ontario (Virtual Conference)*, 2021. To appear.

15 Anindya De. Pseudorandomness for permutation and regular branching programs. In *IEEE Conference on Computational Complexity*, pages 221–231. IEEE Computer Society, 2011. `doi:10.1109/CCC.2011.23`.

16 Oded Goldreich. A sample of samplers - a computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(20), 1997. URL: `http://eccc.hpi-web.de/eccc-reports/1997/TR97-020/index.html`.

17 Oded Goldreich. *A primer on pseudorandom generators*, volume 55 of *University Lecture Series*. American Mathematical Society, Providence, RI, 2010.

18 Oded Goldreich, Salil Vadhan, and Avi Wigderson. Simplified derandomization of bpp using a hitting set generator. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay of Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 59–67. Springer, 2011.

19 Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil Vadhan. Better pseudorandom generators via milder pseudorandom restrictions. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '12)*. IEEE, 20–23 October 2012.

20 William M. Hoza, Edward Pyne, and Salil P. Vadhan. Pseudorandom generators for unbounded-width permutation branching programs. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.7.

21 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 356–364, Montréal, Québec, Canada, 23–25 May 1994.

22 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. Revisiting norm estimation in data streams. *CoRR*, abs/0811.3648, 2008. arXiv:0811.3648.

23 Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products: extended abstract. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 263–272. ACM, 2011. doi:10.1145/1993636.1993672.

24 Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 626–637. ACM, 2019.

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

26 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

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

28 Edward Pyne and Salil Vadhan. Pseudodistributions that beat all pseudorandom generators. ECCC preprint TR21-019, 2021.

29 Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks in regular digraphs and the RL vs. L problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 457–466, 21–23 May 2006. Preliminary version as *ECCC* TR05-22, February 2005.

30 Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM '05)*, number 3624 in Lecture Notes in Computer Science, pages 436–447, Berkeley, CA, August 2005. Springer.

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

32 Jirí Síma and Stanislav Zák. Almost k-wise independent sets establish hitting sets for width-3 1-branching programs. In Alexander S. Kulikov and Nikolay K. Vereshchagin, editors, *CSR*, volume 6651 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2011. doi:10.1007/978-3-642-20712-9_10.

33 Thomas Steinke. Pseudorandomness for permutation branching programs without the group theory. Technical Report TR12-083, Electronic Colloquium on Computational Complexity (ECCC), July 2012. URL: http://eccc.hpi-web.de/report/2012/083/.

34 Salil P Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

35 Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

36 David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.