Error Reduction For Weighted PRGs Against Read Once Branching Programs

Gil Cohen
School of Computer Science, Tel Aviv University, Israel

Dean Doron
Department of Computer Science, Stanford University, USA

Oren Renard
School of Computer Science, Tel Aviv University, Israel

Ori Sberlo
School of Computer Science, Tel Aviv University, Israel

Amnon Ta-Shma
School of Computer Science, Tel Aviv University, Israel

Abstract

Weighted pseudorandom generators (WPRGs), introduced by Braverman, Cohen and Garg [5], are a generalization of pseudorandom generators (PRGs) in which arbitrary real weights are considered, rather than a probability mass. Braverman et al. constructed WPRGs against read once branching programs (ROBPs) with near-optimal dependence on the error parameter. Chattopadhyay and Liao [6] somewhat simplified the technically involved BCG construction, also obtaining some improvement in parameters.

In this work we devise an error reduction procedure for PRGs against ROBPs. More precisely, our procedure transforms any PRG against length n width w ROBP with error $1/\text{poly}(n)$ having seed length s to a WPRG with seed length $s + O(\log \frac{w}{\varepsilon} \cdot \log\log \frac{1}{\varepsilon})$. By instantiating our procedure with Nisan's PRG [17] we obtain a WPRG with seed length $O(\log n \cdot \log(nw) + \log \frac{w}{\varepsilon} \cdot \log\log \frac{1}{\varepsilon})$. This improves upon [5] and is incomparable with [6].

Our construction is significantly simpler on the technical side and is conceptually cleaner. Another advantage of our construction is its low space complexity $O(\log nw) + \text{poly}(\log\log \frac{1}{\varepsilon})$ which is logarithmic in n for interesting values of the error parameter ε. Previous constructions (like [5, 6]) specify the seed length but not the space complexity, though it is plausible they can also achieve such (or close) space complexity.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Pseudorandom generators, Read once branching programs, Space-bounded computation

Digital Object Identifier 10.4230/LIPIcs.CCC.2021.22

Funding *Gil Cohen*: Supported by ERC starting grant 949499 and by Israel Science Foundation grant 1569/18.
Dean Doron: Supported by NSF award CCF-1763311.
Oren Renard: Supported by the Azrieli Faculty Fellowship.
Ori Sberlo: Supported by ERC starting grant 949499 and by ISF grant 952/18.
Amnon Ta-Shma: Supported by Israel Science Foundation grant 952/18.

1 Introduction

1.1 A brief account of space-bounded derandomization

Understanding the role that randomness plays in computation is of central importance in complexity theory. While randomness is provably necessary in many computational settings such as cryptography, PCPs and distributed computing, it is widely believed that randomness adds no significant computational power to neither time- nor space-bounded algorithms. Remarkably, proving such a statement for time-bounded algorithms implies circuit lower bounds which seem to be out of reach of current proof techniques [19, 14, 16].

On the other hand, there is no known barrier for proving such a statement in the space-bounded setting. Indeed, while we cannot even rule out a scenario in which randomness "buys" exponential time, the space-bounded setting is much better understood. Savitch's theorem [23] already implies that any one-sided error randomized algorithm can be simulated deterministically with only a quadratic overhead in space, namely $\mathbf{RL} \subseteq \mathbf{L}^2$. The (possibly) stronger inclusion $\mathbf{BPL} \subseteq \mathbf{L}^2$ can be proven easily through a variant of Savitch's theorem and also follows from [4]. Using pseudorandom generators, Nisan [17, 18] devised a time-efficient derandomization with quadratic overhead in space, concretely, $\mathbf{BPL} \subseteq \mathbf{DTISP}(\mathrm{poly}(n), \log^2 n)$. Focusing solely on space, the state of the art result was obtained by Saks and Zhou [22] that build on Nisan's work to deterministically simulate two-sided error space s randomized algorithms in space $O(s^{3/2})$, thus, establishing that $\mathbf{BPL} \subseteq \mathbf{L}^{3/2}$.

1.2 Pseudorandom generators for ROBPs

Space-bounded algorithms are typically studied by considering their non-uniform counterparts. A length n, width w *read-once branching program* (ROBP) is a directed graph whose nodes, called states, are partitioned to $n + 1$ layers, each consists of at most w states. The first layer contains a designated "start" state, and the last layer consists of two states labeled 'accept' and 'reject'. From every state but for the latter two, there are two outgoing edges, labeled by 0 and 1, to the following layer.[1] On input $x \in \{0, 1\}^n$, the computation proceeds by following the edges according to the labels given by the bits of x starting from the start state. The string x is accepted by the program if the computation ends in the accept state.

A well-known fact (see, e.g., [10, Chapter 5], and [3, Chapter 14.4.4]) is that any space s randomized algorithm in the Turing model can be simulated by a length n, width w ROBP with $n, w = 2^{O(s)}$. Thus, one approach to derandomize two-sided error space-bounded algorithms is to construct, in bounded space, a distribution of small support that "looks random" to any such ROBP. We say that a distribution $\mathcal{D}$ on n-bit strings is (n, w, ε) *pseudorandom* if for every length n, width w ROBP, the path induced by an instruction sequence that is sampled from $\mathcal{D}$ has, up to an additive error ε, the same probability to end in the accept state as a truly random path. A truly random path corresponds to a path picked uniformly at random from the 2^n possible paths. An (n, w, ε) *pseudorandom generator* (PRG) is an algorithm $\mathsf{PRG}\colon \{0, 1\}^s \to \{0, 1\}^n$ that when fed with s uniformly random bits has an output distribution that is (n, w, ε) pseudorandom. We refer to the input to PRG as the *seed*.

[1] For simplicity, here we only consider ROBPs with two outgoing edges. Larger out-degrees (or alphabet) can also be considered and is in fact crucial for obtaining our result even if one is only interested in the binary case.

Derandomizing using a PRG is straightforward. By iterating over all seeds and generating the corresponding instruction sequences, one can calculate the fraction of those paths that end in the accept state. This way, one obtains an ε-approximation to the probability of reaching the accept state while taking a truly random path in the program. The space overhead consists of the seed length s (as an iterator is maintained) and the space of the PRG.

One can prove the existence of an (n, w, ε) PRG with seed length $O(\log(nw/\varepsilon))$. The proof is via the probabilistic method and has no guarantee on the space complexity of the PRG. As such, it is not useful for the purpose of derandomization. In his seminal work, Nisan [17] devised a PRG with seed length $s = O(\log n \cdot \log(nw/\varepsilon))$ and space complexity $O(\log(nw/\varepsilon))$. Setting $n, w = 2^{\Theta(s)}$ and ε to a small constant, the seed length is $O(s^2)$ indeed yields derandomization with quadratic overhead in space. Saks and Zhou [22] applied Nisan's generator in a far more sophisticated way than the naïve derandomization, in particular exploiting its low space complexity, so to obtain their result.

1.3 Pseudorandom pseudo-distributions for ROBPs

Braverman et al. [5] introduced the notion of a *pseudorandom pseudo-distribution* (PRPD) generalizing pseudorandom distributions.

▶ **Definition 1** (pseudorandom pseudo-distribution). *Let* $\rho_1, \ldots, \rho_{2^s} \in \mathbb{R}$ *and* $p_1, \ldots, p_{2^s} \in \{0,1\}^n$. *The sequence* $\widetilde{\mathcal{D}} = ((\rho_1, p_1), \ldots, (\rho_{2^s}, p_{2^s}))$ *is an* (n, w, ε) pseudorandom pseudo-distribution *(PRPD) if for every length n, width w ROBP, the sum of all ρ_i-s for which the respective paths p_i end in the accept state is an ε-approximation to the probability of ending at the accept state by taking a truly random path in the program.*

Note that Definition 1 allows the weights ρ_i to take both positive and negative values. These values are not necessarily bounded by 1 in absolute value, nor by any constant for that matter, and they do not necessarily sum up to 1. Nevertheless, the definition requires that the numbers cancel out nicely so that summing the weights of the respective paths that arrive to the accept state yields an ε-approximation for the probability of arriving to the accept state by taking a truly random path (and, in particular, the sum is a number in $[-\varepsilon, 1+\varepsilon]$). Analogous to a PRG, an (n, w, ε) *weighted pseudorandom generator* (WRPG) is an algorithm $\mathsf{WPRG} \colon \{0,1\}^s \to \mathbb{R} \times \{0,1\}^n$ whose output, when fed with a uniform seed, is an (n, w, ε) PRPD.

A WPRG that can be computed in bounded space suffices to derandomize two-sided error randomized algorithms. Indeed, the straightforward derandomization using a pseudorandom (proper) distribution, which sums the probability mass of the relevant paths, works just as well for pseudo-distributions as one can sum up the weights ρ_i which, in a sense, generalize the probability mass. Of course, the space requirement now depends on the bit complexity of the weights as well.

1.4 The error parameter

Braverman et al. [5] constructed a WPRG that has seed length with an improved–in fact near-optimal–dependence on the error parameter ε. Their WPRG has seed length $O(\log^2 n \cdot \log\log_n \frac{1}{\varepsilon} + \log n \cdot \log w + \log \frac{w}{\varepsilon} \cdot \log\log \frac{w}{\varepsilon})$. For the purpose of derandomization, the error parameter is anyhow taken to be constant, and so the necessity of such an improvement may seem moot. However, by inspecting Nisan's recursive construction one can see that the $\log^2 n$ term in the seed length appears due to the way the error evolves throughout the recursion. Hence, a construction which allows for a more delicate error analysis is called for. Furthermore, the Saks-Zhou construction applies Nisan's PRG in a setting in which $\varepsilon \ll 1/n$ for obtaining their result. It was observed [5] that improving upon [22] can be obtained by constructing a PRG having seed length with better dependence on both w, ε, even when retaining the $\log^2 n$ dependence.

Interestingly (and unfortunately), the $\log^2 n$ term in the BCG construction appears for a completely different reason. In short, unlike prior works [17, 15] that maintain a list of instructions throughout the recursion, BCG maintains a more involved structure consisting of several lists of lists. Maintaining the invariant on this complex structure is the reason for the $\log^2 n$ term in the seed of BCG's construction.

As hinted above, the BCG construction is quite involved. In a subsequent work Chattopadhyay and Liao [6] somewhat simplified the BCG construction also obtaining slight improvement in parameters. In particular, the seed length obtained by [6] is $O(\log n \cdot \log nw \cdot \log\log\log nw + \log \frac{1}{\varepsilon})$. Additionally, Hoza and Zuckerman [13] obtained a significantly simpler construction of hitting sets against ROBPs. Their construction has seed length $O(\frac{1}{\max(1, \log\log w - \log\log n)} \cdot \log n \cdot \log nw + \log \frac{1}{\varepsilon})$. Although hitting sets are weaker objects than PRPDs that are aimed for the derandomization of one sided error randomized algorithms, a subsequent work by Cheng and Hoza [7] showed how to derandomize two sided error randomized algorithms using hitting sets. While this is an illuminating result, we stress that most known constructions of PRGs, WPRGs and hitting sets make use of compositions (either directly or indirectly) and HSGs do not compose well, and so it is very much desired to devise new techniques for constructing PRGs and WPRGs.

1.5 Our contribution

This work further focuses on the error parameter of PRPDs. As our main result, we obtain an *error reduction procedure*. That is, we devise an algorithm that transforms, in a black-box manner, a PRG with a modest error parameter ε_0 to a WPRG with a desired error parameter ε, having comparable seed length and with a near optimal dependence on ε.

▶ **Theorem 2** (main result, see also Corollary 15). *Suppose* PRG *is an* $\left(n, w, n^{-2}\right)$ *PRG with seed length* s_0*, computable in space* m*. Then, for every* ε *there exists an* (n, w, ε) *WPRG with seed length*

$$s = s_0 + O\left(\log \frac{w}{\varepsilon} \cdot \log\log_n \frac{1}{\varepsilon}\right).$$

that is computable in space $O(m + (\log\log \frac{w}{\varepsilon})^3)$.

When instantiated with Nisan's PRG [17] our error reduction procedure yields WPRGs with a seed that is slightly shorter than [5] and is incomparable to [6].

▶ **Corollary 3** (see also Corollary 16). *There exists an* (n, w, ε) *WPRG with seed length*

$$O\left(\log n \cdot \log nw + \log \frac{w}{\varepsilon} \cdot \log\log_n \frac{1}{\varepsilon}\right)$$

computable in space $O\left(\log nw + \left(\log\log \frac{w}{\varepsilon}\right)^3\right)$.

Our error reduction procedure as well as the resulting WPRG are significantly simpler than [5, 6]. Moreover, the underlying ideas are different and conceptually cleaner. More generally, it is much preferred to have a black-box error reduction procedure rather than a specific explicit construction. On top of the insights obtained, such a modularization has the potential of being instantiated in different settings such as for regular and permutation ROBPs or for bounded-width ROBPs.

Our error reduction procedure borrows ideas from the line of work concerning deterministic space-efficient graph algorithms, in particular a recent work by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford and Vadhan [1] (which, in turn, is based on an exciting line of work on nearly-linear time graph algorithms, deterministic or otherwise. See [9, 8] and references therein).

Independently, Pyne and Vadhan [20] also used the Richardson iteration to obtain a WPRG for polynomial-width branching programs, and furthermore used that to obtain new results for permutation BPs.

1.6 An overview of our construction

Let $\mathsf{PRG}\colon \{0,1\}^s \to \{0,1\}^n$ be an (n, w, ε_0) PRG whose error we wish to reduce. Let $\overline{A} = (A_1, \dots, A_n)$ be the $w \times w$ stochastic matrices that correspond to a length n width w ROBP. That is, $A_i = \frac{1}{2}(A_i^{(0)} + A_i^{(1)})$ where $A_i^{(0)}$ is the Boolean stochastic matrix that encodes the edges leaving layer i that are labeled with 0 and $A_i^{(1)}$ encodes the edges labeled with 1. Define the $(n+1)w \times (n+1)w$ lower triangular block matrix B as follows. For $a, b \in [n+1]$, $a > b$, and $\sigma \in \{0,1\}^s$, let

$$B[a,b] = \mathop{\mathbb{E}}_{\sigma \in \{0,1\}^s} \left[A_a^{(\mathsf{PRG}(\sigma)_{a-b})} \cdots A_b^{(\mathsf{PRG}(\sigma)_1)} \right].$$

Further, $B[a,a] = I_w$. Since PRG has error ε_0, for every block $B[a,b]$ with $a > b$, $\|B[a,b] - A_a \cdots A_b\| \le \varepsilon_0$. Following [1] we observe that by denoting

$$L = \begin{pmatrix} I & 0 & \dots & 0 & 0 \\ -A_1 & I & \dots & 0 & 0 \\ 0 & -A_2 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -A_n & I \end{pmatrix},$$

one has that

$$L^{-1} = \begin{pmatrix} I & 0 & \dots & 0 & 0 \\ A_1 & I & \dots & 0 & 0 \\ A_2A_1 & A_2 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_n \dots A_1 & A_n \dots A_2 & \dots & A_n & I \end{pmatrix}.$$

Thus, $\|B - L^{-1}\| \le (n+1)\varepsilon_0$. That is, the crude error PRG can be used to approximate L^{-1} by applying it to all subprograms of the original ROBP.

Richardson iteration is a method for improving a given approximation to an inverse of a matrix. This method is frequently used to construct a preconditioner to a Laplacian system. To describe this method, let $L = I - A$. For $k \ge 1$ define the matrix

$$R_k = \sum_{i=0}^{k} (I - BL)^i B. \tag{1}$$

It can be shown that $\left\|R_k - L^{-1}\right\| \leq (n+1)\left(2(n+1)\varepsilon_0\right)^{k+1}$. Thus, by taking $\varepsilon_0 = n^{-2}$ and $k = O(\log_n \frac{1}{\varepsilon})$, one obtains approximation $\|R_k - L^{-1}\| \leq \varepsilon$. In particular, the lower left block of R_k is an ε-approximation of the desired product $A_n \cdots A_1$.

We further develop Equation (1). Let $\Delta = I - BL$. One can show that

$$\Delta[a,b] = \begin{cases} B[a,b+1]\cdot A_b - B[a,b] & a > b, \\ 0 & a \leq b. \end{cases} \tag{2}$$

Substituting this back to R_k, for $a > b$ we have that

$$R_k[a,b] \;=\; B[a,b] + \sum_{i=1}^{k} \sum_{a>\ell_i>\cdots>\ell_1\geq b} \Delta[a,\ell_i]\cdot\Delta[\ell_i,\ell_{i-1}]\cdots\Delta[\ell_2,\ell_1]\cdot B[\ell_1,b].$$

If we further let $C_0[a,b] = B[a,b+1]\cdot A_b$ and $C_1[a,b] = B[a,b]$ then

$$R_k[a,b] = B[a,b] + \sum_{i=1}^{k} \sum_{a>\ell_1>\cdots>\ell_i\geq b} \sum_{t_1,\ldots,t_i\in\{0,1\}} (-1)^{t_1+\cdots+t_i}\cdot C_{t_i}[a,\ell_i]\cdots C_{t_1}[\ell_2,\ell_1]\cdot B[\ell_1,b]. \tag{3}$$

By extending the definition of ROBPs to arbitrary alphabets (rather than binary) we observe that each summand in Equation (3) can be realized by a ROBP. Our construction thus uses an auxiliary PRG that ε' fools each summand and hence $\varepsilon' n^{O(k)} \approx \varepsilon'\cdot \mathrm{poly}(\frac{1}{\varepsilon})$ approximates R_k which, in turn, ε approximates L^{-1} yielding overall an $O(\varepsilon)$ approximation. As the ROBP that correspond to each summand is short (recall $i \leq k = O(\log_n \frac{1}{\varepsilon}) \ll n$), a short seed is sufficient even for the high accuracy $\varepsilon' = \mathrm{poly}(\varepsilon)$ that we require. We invoke [15] as our auxiliary PRG as it has good dependence on the alphabet size which, in our case, is comparable to the seed of the crude PRG that we started with. We remark that the weights in our PRPD are used so to mimic Equation (3). Indeed, on top of the sign, there are $\binom{n}{i}$ summands that correspond to partition to $i+1$ segments and so the weights are used for creating the appropriate scaling between different values of i.

Discussion.

While $C_1[a,b] = B[a,b]$ is obtained by PRG, $C_0[a,b]$ is computed by following the instructions of PRG for all but the first step. For the latter, we use a fresh random bit. Namely, consider a thought experiment in which we use a new–more expensive–PRG $\mathsf{PRG}'\colon \{0,1\}^{s+1} \to \{0,1\}^{\ell}$ that is defined by $\mathsf{PRG}'(\sigma,p) = p \circ \mathsf{PRG}(\sigma)_{[1,\ell-1]}$, where $\sigma\colon \{0,1\}^s$ and $p \in \{0,1\}$. The matrix $\Delta[a,b] = C_1[a,b] - C_0[a,b]$ then compares the better approximation $C_1[a,b]$ with the "actual" approximation $C_0[a,b]$. From this perspective, Equation (3) suggests interpreting the Richardson iteration as a linear combination with ± 1 coefficients (as determined by $(-1)^{t_1+\cdots+t_i}$) of approximations of $A_n\cdots A_1$ where each approximation is partition to segments (encoded by $\ell_1 > \cdots > \ell_i$). In segment j, according to the value t_j, the relevant sequence of instructions is obtained either from the original PRG or via the refined one PRG'.

1.7 A comparison with [5]

It is worthwhile to explore the differences between the BCG construction [5] (and the followup work of Chattopadhyay and Liao [6] which uses similar ideas) and ours and to point out the aspects of our work that we find similar to the work of Cheng and Hoza [7], and of Hoza and Zuckerman [13]. We start by giving a brief overview of the BCG construction.

1.7.1 A brief overview of BCG

In constructions prior to [5] (e.g., [17, 15]), a list of instructions is maintained with the property that given a ROBP $A_1, \ldots, A_n$, averaging over the products corresponding to the instructions yields the desired approximation to the product $A_n \cdots A_1$. The key idea suggested in [5] is to maintain not a single list whose average yields the desired approximation but rather several lists of instructions $L_0, L_1, \ldots, L_k$ such that averaging according to the instructions in L_0 yields a modest approximation; averaging according to $L_0 \cup L_1$ yields a more refined approximation, and so forth. Averaging according to the instructions given by $L_0 \cup \cdots \cup L_k$ gives the desired approximation. Thus, L_0 can be thought of as a crude approximation, L_1 a first order correction term, L_2 a second order correction term, etc.

To implement this idea, weights were introduced and, moreover, each list but for L_0 was in itself a list of lists, or bundles. The different instructions in a bundle did not carry useful information by themselves and it is the bundle which has the desired properties. Lists that correspond to higher error terms requires the expensive use of bigger bundles and larger weights, and so a delicate use of balanced and unbalanced samplers is employed in [5] in order to maintain the desired invariant throughout the recursion and assuring that the bundles and weights do not get too large.

1.7.2 Comparison with BCG

Our work, in comparison, goes back to the use of a single list as in [17, 15]. We do not need to maintain several lists, let alone lists of bundles. This makes our construction significantly simpler and, in particular, spares us from the delicate application of different types of samplers. The only component we do need are weights, both positive and negative that are unbounded in absolute value. However, it is straightforward to pinpoint the weights used by our construction (see Equation (11)) whereas in [5] the weights are computed via a recursive algorithm. As a result, it is difficult to argue about them. We believe that the simpler and more explicit structure of our construction would enable future works to combine our construction with other ideas for the purpose of obtaining improved constructions and derandomization results.

The common theme to both our construction and BCG is working with cancellations. We "read off" the Richardson iteration what cancellations to consider. As we discussed in the end of Section 1.6, we interpret Richardson iteration as comparing a PRG with the PRG obtained by replacing the first bit by a fresh truly random bit. The BCG construction, on the other hand, "plants" cancellations by considering two samplers–one more refined than the other–and encode their difference in their lists (this requires the introduction of bundles). So, in a sense, BCG's cancellations are obtained by comparing one approximation to another where both approximations are obtained via samplers whereas we make use of one approximation coming from a PRG and another that is obtained by replacing the first bit by a fresh truly uniform bit. The way we combine these is dictated by Richardson iteration.

1.7.3 Common aspects with [13, 7]

For their derandomization result, Cheng and Hoza [7] introduce the notion of *local consistency*. Informally, the authors consider the difference between applying a generated sequence of instructions (via a hitting set) to that obtained by the generated sequence when replacing the last bit with a fresh truly random bit. This is somewhat reminisce to the way we read the cancellations of the Richardson iteration. However, while local consistency is used for making decisions once a ROBP is given, we combine the analog sequences using the Richardson iterator in a block-box matter.

The construction of Hoza and Zuckerman [13] also shares similar aspects with ours. There, they start with a modest-error PRG to get an ε-error hitting set by running the PRG for $k = \log_n(1/\varepsilon)$ times according to partitions of $[n]$ to k segments, resembling what we do. Instead of drawing the PRG's seeds uniformly at random, they derandomize the construction using a hitter. We note however, that their analysis is very different from ours, and uses a progress measure concerning the probability of reaching an accepting state.

2 Preliminaries

2.1 Matrices, branching programs, and space complexity

A matrix is Boolean if all its entries are in $\{0,1\}$, and stochastic if all its entries are nonnegative and the sum of each column is 1. Denote by $\mathrm{BSto}(w)$ the set of $w \times w$ boolean stochastic matrices. We will denote by $\|\cdot\|$ the induced ℓ_1 norm, i.e., $\|A\| = \max_j \sum_i |A_{i,j}|$.

We will often work with block matrices. For instance, we may interpret $A \in \mathbb{R}^{nm \times nm}$ as an $n \times n$ matrix with entries which are $m \times m$ matrices. Whenever this interpretation is clear, we let $A[i,j]$ be the (i,j)-th block. In this example, $A[i,j] \in \mathbb{R}^{m \times m}$.

▶ **Definition 4** (branching program). *Let Σ be some alphabet and let $n, w \in \mathbb{N}$. An (n, Σ, w) branching program (BP) is a sequence $\overline{B} = (B_1, \dots, B_n)$, where each $B_i \colon \Sigma \to \mathrm{BSto}(w)$.*

For $b \le a$ we let $B_{[b,a]}$ be the $(a-b+1, \Sigma, w)$ BP $(B_a, \dots, B_b)$.

▶ **Definition 5.** *The* value *of an (n, Σ, w) BP $\overline{B} = (B_1, \dots, B_n)$ on $x = (x_1, \dots, x_n) \in \Sigma^n$, denoted $\mathrm{val}(\overline{B}, x)$, is the realized $w \times w$ matrix of $\overline{B}$ when fed by x, i.e.*

$$\mathrm{val}\big(\overline{B}, x\big) = B_n(x_n) \cdot B_{n-1}(x_{n-1}) \cdots B_1(x_1).$$

If $\overline{B}$ is the empty sequence, we set $\mathrm{val}(\emptyset, x) = I_w$.

▶ **Definition 6** (weighted PRG). *We say W is an $(n, \Sigma, w, \varepsilon)$-WPRG against BPs with seed length s if:*

- *$W = (I, \mu)$ where $I \colon \{0,1\}^s \to \Sigma^n$ and $\mu \colon \{0,1\}^s \to \mathbb{R}$, and,*
- *For every (n, Σ, w) BP $\overline{B} = (B_1, \dots, B_n)$, it holds that*

$$\left\| \mathop{\mathbb{E}}_{x \in \{0,1\}^s}\left[\mu(x) \cdot \mathrm{val}\big(\overline{B}, I(x)\big)\right] - \mathop{\mathbb{E}}_{x \in \Sigma^n}\left[\mathrm{val}\big(\overline{B}, x\big)\right] \right\| \le \varepsilon.$$

When $\mu \equiv 1$, we say that W is a PRG.

For $1 \le \ell \le n$ we let $G_\ell \colon \{0,1\}^{s_0} \to \Sigma^\ell$ be the first ℓ symbols of the output of G. Note that if $G \colon \{0,1\}^{s_0} \to \Sigma^n$ is an $(n, \Sigma, w, \varepsilon)$ PRG then G_ℓ is an $(\ell, \Sigma, w, \varepsilon)$ PRG.

We say $f : \Lambda_1 \to \Lambda_2$ is computable in space s, if given $x \in \Lambda_1$ and index j, $f(x)_j \in \Lambda_2$ can be computed in additional work space that consists of s bits. We will use the following well known theorem regarding the space complexity of compositions.

▶ **Theorem 7.** *Let $f_1, f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$ be two functions that can be computed in $s_1, s_2 \colon \mathbb{N} \to \mathbb{N}$ space such that $s_1(n), s_2(n) = \Omega(\log n)$. Then, on input x, $f_2 \circ f_1 : \{0,1\}^\star \to \{0,1\}^\star$ can be computed using $O(s_1(|x|) + s_2(|f_1(x)|))$ space.*

2.2 Known PRG constructions

▶ **Theorem 8** ([17, 18]). *For any positive integers n, w, any error parameter $\varepsilon > 0$ and any alphabet Σ, there exists an $(n, \Sigma, w, \varepsilon)$ PRG with seed length*

$$s \;=\; O\left(\log n \cdot \log \frac{nw|\Sigma|}{\varepsilon}\right),$$

computable in space $\min\left\{O\left(\log \frac{nw|\Sigma|}{\varepsilon}\right), O\left(\log n \cdot \log\log \frac{nw|\Sigma|}{\varepsilon}\right)\right\}$.

▶ **Theorem 9** ([15]). *For any positive integers n, w, any error parameter $\varepsilon > 0$ and any alphabet Σ, there exists an $(n, \Sigma, w, \varepsilon)$ PRG with seed length*

$$s \;=\; \log|\Sigma| + O\left(\log n \cdot \log\left(\frac{nw}{\varepsilon}\right)\right),$$

computable in space $O\left(\log n \cdot \left(\log\log \frac{nw|\Sigma|}{\varepsilon}\right)^2\right)$.

Theorem 8 is derived almost directly from [17, 18], and Theorem 9 follows from [15], except for the space complexity which is implicit in those works and also depends on the specific implementation. For completeness, we give the proof of Theorem 8 in Appendix B.1, and of Theorem 9 in Appendix B.3.

3 Richardson iteration

Let A be an invertible $n \times n$ real matrix, and assume that B approximates A^{-1}, concretely, $\|B - A^{-1}\| \le \varepsilon_0$ for some sub-multiplicative norm. Richardson iteration is a method for obtaining a more refined approximation of A^{-1} given access to the crude B as well as to the original matrix A.

▶ **Lemma 10.** *Let $L \in \mathbb{R}^{m\times m}$ be an invertible matrix and $A \in \mathbb{R}^{m\times m}$ such that $\left\|L^{-1} - A\right\| \le \varepsilon_0$. For any nonnegative integer k, define*

$$\mathrm{R}(A, L, k) = \sum_{i=0}^{k} (I - AL)^i A.$$

Then, $\left\|L^{-1} - \mathrm{R}(A, L, k)\right\| \le \left\|L^{-1}\right\| \cdot \|L\|^{k+1} \cdot \varepsilon_0^{k+1}$.

The proof is deferred to Appendix A.

Following [1] we will be interested in the following instantiation of the Richardson iteration. Let $\overline{M} = (M_1, \dots, M_n)$ be a sequence of $w \times w$ matrices. We consider the $(n+1)w \times (n+1)w$ matrix

$$M = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ M_1 & 0 & \dots & 0 & 0 \\ 0 & M_2 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_n & 0 \end{pmatrix}. \tag{4}$$

The Laplacian of M is $L = I_{(n+1)w} - M$, and we treat L as an $(n+1) \times (n+1)$ block matrix. The following claim follows by a simple calculation.

▷ Claim 11. For $i, j \in [n+1]$, the (i,j)-th block of L^{-1} is given by

$$L^{-1}[i,j] = \begin{cases} M_{i-1} \cdots M_j & i > j, \\ I_w & i = j, \\ 0 & i < j. \end{cases}$$

Richardson for branching programs.

Let $\overline{B} = (B_1, \ldots, B_n)$ be an (n, Σ, w) BP and let $M_i = \mathbb{E}_{\sigma \in \Sigma}[B_i(\sigma)]$ be the corresponding transition matrices. Thus, approximating the transition probabilities of $\overline{B}$,

$$\mathop{\mathbb{E}}_{x \in \Sigma^n}\left[\mathrm{val}\big(\overline{B}, x\big)\right] = M_n \cdots M_1,$$

amounts to approximating the lowest leftmost entry $L^{-1}[n+1, 1]$.

▷ Claim 12. Let $\overline{B} = (B_1, \ldots, B_n)$ be an (n, Σ, w) BP. Set $M_i = \mathbb{E}_{\sigma \in \Sigma}[B_i(\sigma)]$ and L as in Equation (4). Also, let $G\colon \{0,1\}^s \to \Sigma^n$ be an $(n, \Sigma, w, \varepsilon_0)$ PRG and consider

$$A[a,b] = \begin{cases} \mathbb{E}_{x \in \{0,1\}^s}\left[\mathrm{val}\big(B_{[b,a-1]}, G_{a-b}(x)\big)\right], & a \geq b \\ 0 & a < b. \end{cases} \tag{5}$$

Then,

$$\left\|L^{-1} - \mathrm{R}(A, L, k)\right\| \leq (n+1) \cdot (2\varepsilon_0)^{k+1}.$$

Let A as in Equation (5) and write $\mathrm{R}(A, L, k) = \sum_{i=0}^{k} \Delta^i A$ where $\Delta = I - AL$. Denote $A' = A - I$, i.e., A' is the part of A below the main diagonal. Then,

$$\Delta = I - AL = I - A(I - M) = (I - A) + AM = AM - A'.$$

In block notation, for $a, b \in [n+1]$, following Equation (4),

$$AM[a,b] = \sum_{i=1}^{n+1} A[a,i]M[i,b] = A[a,b+1]M[b+1,b] = A[a,b+1] \cdot M_b.$$

Thus,

$$\Delta[a,b] = \begin{cases} A[a,b+1] \cdot M_b - A[a,b] & a > b, \\ 0 & a \leq b. \end{cases} \tag{6}$$

Going back to $\mathrm{R}(A, L, k)$, for $a > b$ we have that

$$\mathrm{R}(A,L,k)[a,b] = A[a,b] + \sum_{i=1}^{k} \sum_{a > r_i > \cdots > r_1 \geq b} \Delta[a, r_i] \cdot \Delta[r_i, r_{i-1}] \cdots \Delta[r_2, r_1] \cdot A[r_1, b]. \tag{7}$$

If we further let $C_0[a,b] = A[a,b+1] \cdot M_b$ and $C_1[a,b] = A[a,b]$, then

$$\mathrm{R}(A,L,k)[a,b] = A[a,b] + \sum_{\substack{\bar{t} \in \{0,1\}^i \\ a > r_i > \cdots > r_1 \geq b}} \sum_{t_1, \ldots, t_i \in \{0,1\}} (-1)^{t_1 + \cdots + t_i} \cdot C_{t_i}[a, r_i] \cdots C_{t_1}[r_2, r_1] \cdot A[r_1, b]. \tag{8}$$

4 The construction

4.1 Black-box error reduction

Let $G\colon \{0,1\}^{s_0} \to \Sigma^n$ be an $(n, \Sigma, w, \varepsilon_G)$ and $G_{\mathrm{aux}}\colon \{0,1\}^{s_{\mathrm{aux}}} \to (\{0,1\}^{s_0} \times \Sigma)^{k+1}$ be a $(k+1, \{0,1\}^{s_0} \times \Sigma, w, \varepsilon_{\mathrm{aux}})$ PRG. Also, for $t \in \{0,1\}$ and $\sigma \in \Sigma$ we let

$$G_{t,\ell}(x,\sigma) = \begin{cases} \sigma \circ G_{\ell-1}(x) & t = 0, \\ G_\ell(x) & t = 1. \end{cases} \tag{9}$$

We now define the WPRG $(I, \mu)\colon \{0,1\}^s \to \Sigma \times \mathbb{R}$. The seed $x \in \{0,1\}^s$ to our WPRG is interpreted as follows.

- The first $\log(k+1)$ bits encode $i \in \{0, \dots, k\}$.
- The next $\log \binom{n}{i}$ bits encode a sequence $\overline{\ell} = (\ell_0, \ell_1, \dots, \ell_i)$ such that $\ell_0 + \cdots + \ell_i = n$, $\ell_i, \dots, \ell_1 > 0$, and $\ell_0 \ge 0$.
- The next i bits are denoted by $\overline{t} = \overline{t} = (t_1, \dots, t_i) \in \{0,1\}^i$.
- The next s_{aux} bits are denoted by $x_{\mathrm{aux}} \in \{0,1\}^{s_{\mathrm{aux}}}$.

Overall, we can write $x = (i, \overline{\ell}, \overline{t}, x_{\mathrm{aux}})$, and the WPRG (I, μ) has seed length

$$s = s_{\mathrm{aux}} + O(k \log n). \tag{10}$$

For brevity we sometimes omit the dependence of i, $(\ell_0, \dots, \ell_i)$, $(t_1, \dots, t_i)$, and x_{aux} on x. We define I and μ as follows.

$$I(x) = \begin{cases} G_n(G_{\mathrm{aux}}(x_{\mathrm{aux}})_0) & i = 0, \\ G_{t_i,\ell_i}(G_{\mathrm{aux}}(x_{\mathrm{aux}})_i) \circ \cdots \circ G_{t_1,\ell_1}(G_{\mathrm{aux}}(x_{\mathrm{aux}})_1) \circ G_{\ell_0}(G_{\mathrm{aux}}(x_{\mathrm{aux}})_0) & \text{otherwise.} \end{cases}$$

$$\mu(x) = \begin{cases} k+1 & i = 0, \\ (k+1) \cdot \binom{n}{i} \cdot 2^i \cdot (-1)^{t_1 + \cdots + t_i} & \text{otherwise.} \end{cases} \tag{11}$$

where $G_{\mathrm{aux}}(x_{\mathrm{aux}})_j$ denotes the j'th symbol in $G_{\mathrm{aux}}(x_{\mathrm{aux}}) \in (\{0,1\}^{s_0} \times \Sigma)^{k+1}$.

The weights are chosen so that the approximation yielded by the above WPRG is a derandomized version of Equation (8) for $(a, b) = (n+1, 1)$. Note that in Equation (8) we used $r_1, \dots, r_i$ which partitioned the interval $[n+1, 1]$, while in Equation (11) we used $\ell_0, \dots, \ell_i$ that sum to n. This is merely an alternative way of writing the sum – the ℓ_i-s are the sum of differences of the r_i-s.

4.2 Correctness

In this section we use the same notation as in Section 3.

▶ **Lemma 13.** *Let $0 < \varepsilon < \varepsilon_0 = \frac{1}{4n}$ and let $k = \log_{1/\varepsilon_0}(1/\varepsilon)$. Suppose*

- *$G\colon \{0,1\}^{s_0} \to \Sigma^n$ is an $\left(n, \Sigma, w, \varepsilon_G = \frac{\varepsilon_0}{2(n+1)}\right)$ PRG, and,*
- *$G_{\mathrm{aux}}\colon \{0,1\}^{s_{\mathrm{aux}}} \to (\{0,1\}^{s_0} \times \Sigma)^{k+1}$ is a $(k+1, \{0,1\}^{s_0} \times \Sigma, w, \varepsilon_{\mathrm{aux}} = \varepsilon^3)$ PRG.*

Then, (I, μ) is an $(n, \Sigma, w, \varepsilon)$ WPRG with seed length $s = s_{\mathrm{aux}} + O(\log(1/\varepsilon))$ computable in space $O(\mathrm{space}(G_{\mathrm{aux}}) + \mathrm{space}(G) + \log s)$.

Proof. Assume k, G and G_{aux} are as in the hypothesis of the lemma. The space complexity follows from Theorem 7 and the seed length was analyzed in Equation (10). We are left to prove that (I,μ) is an (n,Σ,w,ε) WPRG. Fix any (n,Σ,w) BP $\overline{B}=(B_1,\dots,B_n)$. Let A be the $(n+1)w\times(n+1)w$ lower triangular block matrix in which

$$A[a,b]=\mathop{\mathbb{E}}_{x\in\{0,1\}^{s_0}}\left[\mathrm{val}\big(B_{[b,a-1]},G_{a-b}(x)\big)\right]$$

for $a>b$, and $A[a,a]=I_w$. Since G is $\left(n,\Sigma,w,\varepsilon_G=\frac{\varepsilon_0}{2(n+1)}\right)$ PRG we have that

$$\left\|L^{-1}[a,b]-A[a,b]\right\|\le\varepsilon_G$$

and $\left\|L^{-1}-A\right\|\le(n+1)\varepsilon_G$. By our choice of μ,

$$\mathop{\mathbb{E}}_{x\in\{0,1\}^{s}}\left[\mu(x)\cdot\mathrm{val}\big(\overline{B},I(x)\big)\right]=\sum_{i=0}^{k}\sum_{\overline{t},\overline{\ell}}(-1)^{t_1+\dots+t_i}\cdot\mathop{\mathbb{E}}_{x_{\text{aux}}}\left[\mathrm{val}\big(\overline{B},I(i,\overline{\ell},\overline{t},x_{\text{aux}})\big)\right],$$

and

$$\mathrm{R}(A,L,k)[n+1,1]=A[n+1,1]+\sum_{i=1}^{k}\sum_{\overline{t},\overline{r}}(-1)^{t_1+\dots+t_i}\cdot C_{t_i}[n+1,r_i]\cdots C_{t_1}[r_2,r_1]\cdot A[r_1,1],$$

where $\ell_0+\dots+\ell_i=n$ and $n+1>r_i>\dots>r_1\ge 1$. We soon prove:

▷ Claim 14. For every fixed $i\in\{0,\dots,k\}$, $\overline{t}\in\{0,1\}^i$, and $\overline{\ell}$ such that $\ell_0+\dots+\ell_i=n$

$$\left\|\mathop{\mathbb{E}}_{x_{\text{aux}}}\left[\mathrm{val}\big(\overline{B},I(i,\overline{\ell},\overline{t},x_{\text{aux}})\big)\right]-C_{t_i}[n+1,r_i]\cdots C_{t_1}[r_2,r_1]\cdot A[r_1,1]\right\|\ \le\ \varepsilon_{\text{aux}},$$

where $r_j=1+\ell_0+\dots+\ell_{j-1}$.

As we have at most $(k+1)n^k2^k$ summands, we see that

$$\begin{aligned}\left\|\mathop{\mathbb{E}}_{x\in\{0,1\}^{s}}\left[\mu(x)\cdot\mathrm{val}\big(\overline{B},I(x)\big)\right]-\mathrm{R}(A,L,k)[n+1,1]\right\|&\le(k+1)n^k2^k\cdot\varepsilon_{\text{aux}}\\&\le\frac{n^{2k}}{2}\cdot\varepsilon_{\text{aux}}\le\frac{\varepsilon}{2}.\end{aligned}$$

It therefore follows from Claim 12 that

$$\begin{aligned}\left\|\mathrm{R}(A,L,k)[n+1,1]-\mathop{\mathbb{E}}_{x\in\Sigma^n}\left[\mathrm{val}\big(\overline{B},x\big)\right]\right\|&\le(n+1)\big(2(n+1)\varepsilon_G\big)^{k+1}\\&\le 2n\cdot\varepsilon_0^{k+1}\le 2n\varepsilon_0\varepsilon=\frac{\varepsilon}{2},\end{aligned}$$

which together completes the proof. ◀

Proof of Claim 14. Fix $i\in\{0,\dots,k\}$, $\ell_0+\dots+\ell_i=n$, and $\overline{t}\in\{0,1\}^i$ and recall that $r_j=1+\ell_0+\dots+\ell_{j-1}$. We define a $(k+1,\{0,1\}^{s_0}\times\Sigma,w)$ BP $\overline{B'}=(B'_0,\dots,B'_k)$ (that depends on i, $\overline{\ell}$, and $\overline{t}$) such that for all $j=0,\dots,k$,

$$B'_j(x,\sigma)\quad=\quad\begin{cases}\mathrm{val}\big(B_{[r_j,r_{j+1}-1]},\sigma\circ G_{\ell_j-1}(x)\big) & j>0,t=0,\\ \mathrm{val}\big(B_{[r_j,r_{j+1}-1]},G_{\ell_j}(x)\big) & j>0,t=1,\\ \mathrm{val}\big(B_{[1,r_1-1]},G_{\ell_0}(x)\big) & j=0.\end{cases}\tag{12}$$

We stress that B'_j is a BP because a product of Boolean stochastic matrices is Boolean stochastic. The claim now follows since G_{aux} is a $(k+1,\{0,1\}^{s_0}\times\Sigma,w,\varepsilon_{\text{aux}})$ PRG. ◀

4.3 The final construction

We now instantiate Lemma 13 with G_{aux} being the INW PRG from Theorem 9 and G being an arbitrary PRG. The reason for using the INW generator is its additive dependence on $\log|\Sigma|$.

▶ **Corollary 15.** *Let $G\colon \{0,1\}^{s_0} \to \Sigma^n$ be an $(n,\Sigma,w,\varepsilon_G)$. Then, for any error parameter $\frac{1}{4n} > \varepsilon > 0$ there exists an (n,Σ,w,ε) WPRG with seed length*

$$s_0 + O\left(\log\frac{w}{\varepsilon}\cdot\log\log_n\frac{1}{\varepsilon}\right)$$

computable in space $O\Big(\text{space}(G) + \log\log_n(1/\varepsilon)\cdot\big(\log\log\frac{w}{\varepsilon}\big)^2\Big)$.

Had we used Nisan's PRG from Theorem 8 instead of INW then the seed length would deteriorate to

$$O\left(s_0\cdot\log\log_n\frac{1}{\varepsilon} + \log\frac{w}{\varepsilon}\cdot\log\log_n\frac{1}{\varepsilon}\right).$$

Corollary 15 can be interpreted as an error reduction procedure for PRGs with a slight overhead in the seed and space complexity. We proceed by applying this error reduction to Nisan's PRG from Theorem 8.

▶ **Corollary 16.** *For any positive integers n, w, any error parameter $\frac{1}{4n} > \varepsilon > 0$ and any alphabet Σ, there exists an (n,Σ,w,ε) WPRG with seed length*

$$O\left(\log n\log(nw|\Sigma|) + \log\frac{w}{\varepsilon}\cdot\log\log_n\frac{1}{\varepsilon}\right)$$

computable in space $O\Big(\log(nw|\Sigma|) + \log\log_n(1/\varepsilon)\cdot\big(\log\log\frac{w}{\varepsilon}\big)^2\Big)$.

Note that for ε which is not tiny the space complexity is dominated by the first term. Specifically, for $\varepsilon > 2^{-2^{\log^{1/3} n}}$, $w < 2^{2^{\log^{1/3} n}}$ the space complexity is indeed $O(\log(nw|\Sigma|))$. Had we used INW instead, the space complexity would deteriorate to

$$O\left(\log n\cdot\left(\log\log\frac{nw|\Sigma|}{\varepsilon}\right)^2 + \log\frac{w}{\varepsilon}\cdot\log\log_n\frac{1}{\varepsilon}\right).$$

References

1. AmirMahdi Ahmadinejad, Jonathan Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil Vadhan. High-precision estimation of random walks in small space. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1295–1306. IEEE, 2020.
2. Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.
3. Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.
4. Allan Borodin, Stephen Cook, and Nicholas Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1-3):113–136, 1983.
5. Mark Braverman, Gil Cohen, and Sumegha Garg. Pseudorandom pseudo-distributions with near-optimal error for read-once branching programs. *SIAM Journal on Computing*, 49(5):STOC18–242–STOC18–299, 2020.

6 Eshan Chattopadhyay and Jyun-Jie Liao. Optimal error pseudodistributions for read-once branching programs. In *Proceedings of the 35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

7 Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

8 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*. ACM, 2017.

9 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*. IEEE, 2016.

10 Oded Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, Cambridge, 2008.

11 Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

12 Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS 2006)*. Springer, 2006.

13 William M. Hoza and David Zuckerman. Simple optimal hitting sets for small-success **RL**. *SIAM Journal on Computing*, 49(4):811–820, 2020.

14 Russel Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

15 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM SIGACT Symposium on Theory of Computing (STOC 1994)*. ACM, 1994.

16 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity*, 13(1-2):1–46, 2004.

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

18 Noam Nisan. $\mathbf{RL} \subseteq \mathbf{SC}$. *computational complexity*, 4(1):1–11, 1994.

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

20 Edward Pyne and Salil Vadhan. personal communication, February 2021.

21 Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st Annual ACM SIGACT Symposium on Theory of Computing (STOC 1999)*. ACM, 1999.

22 Michael E. Saks and Shiyu Zhou. $\mathsf{BP_H SPACE}(S) \subseteq \mathsf{DSPACE}(S^{2/3})$. *Journal of Computer and System Sceinces*, 58(2):376–403, 1999.

23 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, Apr 1970.

A Proof of Lemma 10

We restate Lemma 10.

▶ **Lemma 17.** *Let $L \in \mathbb{R}^{m\times m}$ be an invertible matrix and $A \in \mathbb{R}^{m\times m}$ such that $\left\|L^{-1} - A\right\| \le \varepsilon_0$. For any nonnegative integer k, define*

$$\mathrm{R}(A, L, k) = \sum_{i=0}^{k}(I - AL)^i A.$$

Then, $\left\|L^{-1} - \mathrm{R}(A, L, k)\right\| \le \left\|L^{-1}\right\| \cdot \|L\|^{k+1} \cdot \varepsilon_0^{k+1}$.

Proof. For any matrix Z, the matrices I and Z commute, and so by a straightforward induction,

$$I - \sum_{i=0}^{k}(I - Z)^i Z = (I - Z)^{k+1}.$$

In particular, for $Z = AL$,

$$I - \mathrm{R}(A, L, k) \cdot L = (I - AL)^{k+1}.$$

Thus,

$$\begin{aligned}\left\|L^{-1} - \mathrm{R}(A, L, k)\right\| &= \left\|(I - \mathrm{R}(A, L, k) \cdot L) \cdot L^{-1}\right\| \\ &\le \left\|L^{-1}\right\| \cdot \|I - \mathrm{R}(A, L, k) \cdot L\| \\ &\le \left\|L^{-1}\right\| \cdot \|I - AL\|^{k+1} \\ &= \left\|L^{-1}\right\| \cdot \left\|(L^{-1} - A) \cdot L\right\|^{k+1} \\ &\le \left\|L^{-1}\right\| \cdot \|L\|^{k+1} \cdot \varepsilon_0^{k+1}.\end{aligned}$$

◀

B The space complexity of some pseudorandom objects

In this section we show how to achieve the space complexity declared in Theorem 8 and Theorem 9. For the INW generator we choose a specific implementation with a small space complexity. The constructions are well known, and the variant of INW we use was explored by [12]. We give it here for completeness.

B.1 Nisan's generator

Proof sketch of Theorem 8. We are given parameters $n, \Sigma, w, \varepsilon$. We set $X = [A]$ for $A = O\left(\frac{nw\Sigma}{\varepsilon}\right)$. We let $\mathcal{H}$ be a 2-universal family of hash functions over X where $|\mathcal{H}| = A^2$ and $h(x)$, for $h \in \mathcal{H}$ and $x \in X$, can be computed in space $O(\log\log|X|)$ (see [17, 18]).

Nisan's generator interprets the seed as $y, h_1, \dots, h_{\log n}$, where $y \in X$, and $h_1, \dots, h_{\log n} \in \mathcal{H}$. For $j \in [n]$, the j-th symbol in the output of the generator is $h_1^{b_1}\left(h_2^{b_2}\left(\cdots h_{\log n}^{b_{\log n}}(y)\right)\right)$, where $(b_1, \dots, b_{\log n}) \in \{0,1\}^{\log n}$ is the binary representation of j, and h^b is either h, if $b = 1$, or the identity function, if $b = 0$. Given $y, h_1, \dots, h_{\log n}$, $j = (b_1, \dots, b_{\log n})$ we can compute the j-th output symbol in the following two alternative ways.

- We can successively compute $h_j^{b_j}\left(\cdots h_{\log n}^{b_{\log n}}(y)\right)$ for $j = \log n, \dots, 1$, each time keeping the current X-symbol. This takes

$$O\left(\log\frac{nw|\Sigma|}{\varepsilon} + \log\log n + \log\log|X|\right) = O\left(\log\frac{nw|\Sigma|}{\varepsilon}\right)$$

space.

- Alternatively, we can do the above computation using composition of space bounded reductions, resulting in space complexity

$$O(\log n \cdot \log\log |X|) = O\left(\log n \cdot \log\log \frac{nw|\Sigma|}{\varepsilon}\right).$$

◀

B.2 A high min-entropy extractor

To apply INW, we need a space-efficient *seeded extractor* with a small entropy loss in the high min-entropy regime. Goldreich and Wigderson [11] gave such a construction utilizing a regular expander $G = (V, E)$ with a small normalized second eigenvalue. For our expander, we choose a Cayley graph over the commutative group $\mathbb{Z}_2^n$ with a generator set $S \subseteq \{0,1\}^n$ that is λ-biased. It is well known that $Cay(\mathbb{Z}_2^n, S)$ has normalized second largest eigenvalue at most λ. For the λ-biased set we choose a construction from [2]. Altogether, this unfolds for the following.

- For the λ-biased set S, first pick q to be the first power of two larger than $\frac{n}{\lambda}$. The set S is of cardinality q^2. For every $\alpha, \beta \in \mathbb{F}_q$ there is an elements $s_{\alpha,\beta} \in \mathbb{Z}_2^n$ where $(s_{\alpha,\beta})_i = \langle \alpha^i, \beta \rangle$, such that multiplication is in $\mathbb{F}_q$ and the inner product is over $\mathbb{Z}_2$. [2] showed the set is λ-biased.
- We let $G = (V, E)$ with $V = \mathbb{Z}_2^n$ and $(x, y) \in E$ iff $x + y \in S$. G is a λ-expander.

The extractor $\mathsf{GW}\colon \{0,1\}^n \times [D] \to \{0,1\}^n$ is defined by letting $G(x, i)$ be the i-th neighbour of x in the graph G.

▷ Claim 18. Let $0 < \Delta < n$ and set G and GW as above. Then, $\mathsf{GW}\colon \{0,1\}^n \times [D] \to \{0,1\}^n$ is a $(k = n - \Delta, \varepsilon)$ extractor with seed length $d = O(\Delta + \log \frac{n}{\varepsilon})$ and space complexity $O(\log n \cdot \log(\Delta + \log(n/\varepsilon)))$.

Proof. For correctness, note that the expander mixing lemma shows that GW is an $(n-\Delta, \varepsilon = O(2^{\Delta/2}\lambda))$ extractor.

Seed length. The seed length of this extractor is $\log |S| = O(\log \frac{n}{\lambda}) = O(\log \frac{n2^\Delta}{\varepsilon}) = O(\Delta + \log \frac{n}{\varepsilon})$.

Space complexity. The space complexity of computing $\mathsf{GW}(x, y)$ given x and y, is the space needed to compute $s_y \in S$ from $y = (\alpha, \beta) \in \mathbb{F}_q^2$, plus the space needed to compute $x + s_y$. The dominating step in computing s_y is computing α^i (for $i \le n$) which can be done in $O(\log n \log\log q)$ with space composition. Altogether, the space needed is $O(\log n \cdot \log\log \frac{n}{\lambda}) = O\left(\log n \cdot \log\log \frac{n2^\Delta}{\varepsilon}\right)$.

We note that Healy and Viola [12] gave an extremely efficient implementation of the above AGHP generator, yielding a better space complexity of $O(\log(n + \log q))$ to compute $\langle \alpha^i, \beta \rangle$. However, in our overall setting of parameters it will make negligible difference.

◀

We remark that by using expanders with better dependence between D and λ, one can get $d = O(\Delta + \log \frac{1}{\varepsilon})$, but here we care more about the space complexity, and $\log n$ factors are negligible for us.

B.3 The INW generator

Proof sketch of Theorem 9. We consider the INW generator [15] instantiated with extractors (as, e.g., in [21]). We are given parameters n, Σ, w, and $\varepsilon = \varepsilon_{\mathsf{INW}}$. We set parameters $\Delta = \log w + O(\log \frac{n}{\varepsilon})$, and d as the seed length for the extractor of Claim 18 for length n, error $\varepsilon_{\mathsf{Ext}} = \frac{\varepsilon}{n}$ and Δ. We let $s = \log|\Sigma| + \log n \cdot 2d$ and we assume $s \le n$. We let $\ell_i = s - i \cdot \Delta$ for $0 \le i \le n$.

Given a seed $x \in \{0,1\}^s$ we view the computation of $\mathsf{INW}(x)$ as a full binary tree of depth $\log n$. Nodes in level i of the tree are labeled by strings of length ℓ_i. The root (at level 0) is labeled by x (of length $\ell_0 = s$). Given any internal node in level $i \in \{0, \dots, \log n\}$ labeled by some string $z \in \{0,1\}^{\ell_i}$, we write $z = z_1 \circ z_2$ with $z_i \in \{0,1\}^{\ell_{i+1}}$ and $z_2 \in \{0,1\}^d$. The left child of z is labeled with z_1, and the right child of z is labeled with $\mathsf{Ext}_i(z_1, z_2)$, where Ext_i is given by Claim 18 for Δ, length ℓ_{i+1} and error $\varepsilon_{\mathsf{Ext}}$ (notice that since $\ell_i < n$, d bits suffice for the seed). $\mathsf{INW}(x)$ is the concatenation of the leaf's labels, from left to right, truncating outputs to $\log|\Sigma|$ bits.

Given an index $j \in [n]$, computing $\mathsf{INW}(x)_j \in \Sigma$ can be done by walking down the computation tree, and each time either truncating a string or invoking an extractor. By composition of space bounded reductions the space complexity of the construction is $\log n$ times the space complexity of the worst extractor used. That is, $\log n \cdot \log \ell_0 \cdot \log(\Delta + \log \frac{\ell_0}{\varepsilon_{\mathsf{Ext}}})$. Plugging-in Δ and $\varepsilon_{\mathsf{Ext}}$, the space complexity is bounded by

$$O\left(\log n \cdot \log \ell_0 \cdot \log\log \frac{nw}{\varepsilon}\right) = O\left(\log n \cdot \log\left(\log|\Sigma| + \log n \log \frac{nw}{\varepsilon}\right) \cdot \log\log \frac{nw}{\varepsilon}\right)$$
$$= O\left(\log n \cdot \left(\log\log \frac{nw|\Sigma|}{\varepsilon}\right)^2\right).$$

◀