Electronic Colloquium on Computational Complexity, Report No. 149 (2022)

Approximating Iterated Multiplication of Stochastic Matrices in Small Space [*]

Gil Cohen[†] Dean Doron[‡] Ori Sberlo[§] Amnon Ta-Shma[¶]

Abstract

Matrix powering, and more generally iterated matrix multiplication, is a fundamental linear algebraic primitive with myriad applications in computer science. Of particular interest is the problem's space complexity as it constitutes the main route towards resolving the **BPL** vs. **L** problem. The seminal work by Saks and Zhou [SZ99] gives a deterministic algorithm for approximating the product of n stochastic matrices of dimension $w \times w$ in space $O(\log^{3/2} n + \sqrt{\log n} \cdot \log w)$. The first improvement upon [SZ99] was achieved by Hoza [Hoz21] who gave a logarithmic improvement in the $n = \mathrm{poly}(w)$ regime, attaining $O(\frac{1}{\sqrt{\log\log n}} \cdot \log^{3/2} n)$ space.

We give the first polynomial improvement over [SZ99]. Our algorithm achieves space complexity of

$$\widetilde{O}\left(\log n + \sqrt{\log n} \cdot \log w\right).$$

In particular, in the regime $\log n > \log^2 w$, our algorithm runs in nearly-optimal $\widetilde{O}(\log n)$ space, improving upon the previous best $O(\log^{3/2} n)$.

To obtain our result for the special case of matrix powering, we harness recent machinery from time- and space-bounded Laplacian solvers to the [SZ99] framework and devise an intricate precision-alternating recursive scheme. This enables us to bypass the bottleneck of paying $\log n$-space per recursion level. The general case of iterated matrix multiplication poses several additional challenges, the substantial of which is handled by devising an improved shift and truncate mechanism. The new mechanism is made possible by a novel use of the Richardson iteration.

[*]This paper subsumes a manuscript authored by the first three authors [CDS22] which dealt only with the case of matrix powering.

[†]Tel Aviv University. gil@tauex.tau.ac.il. Funded by ERC starting grant 949499 and by the Israel Science Foundation grant 1569/18.

[‡]Ben Gurion University. deand@bgu.ac.il.

[§]Tel Aviv University. orisberlo@mail.tau.ac.il. Funded by ERC starting grant 949499 and by ISF grant 952/18.

[¶]Tel Aviv University. amnon@tauex.tau.ac.il. Funded by ISF grant 952/18.

ISSN 1433-8092

Contents

1 Introduction

One of the great open problems of computational complexity is the $\mathbf{BPL}$ vs. $\mathbf{L}$ problem: To what extent is randomness necessary for space-bounded algorithms? More concretely, can every probabilistic algorithm be fully derandomized with only a constant factor blowup in space? The problem withstood countless attempts, even though it is widely believed that $\mathbf{BPL} = \mathbf{L}$ (as indeed follows from plausible circuit lower bounds [KvM02]), and there are no known barriers for the unconditional derandomization of $\mathbf{BPL}$.

The problem of derandomizing $\mathbf{BPL}$ is equivalent to the problem of approximating powers of stochastic matrices. Indeed, a Turing machine M that uses space $S = \log w$ can be converted to a Markov chain A on $O(w)$ states, and vice versa. Assuming M uses n random bits, one is interested in approximating the probability of reaching some accepting configuration t starting from the initial configuration s. This clearly translates to approximating $A^n[s,t]$.

A (halting) $\mathbf{BPL}$ machine is only allowed $\mathrm{poly}(w)$ running time and can toss at most one coin per step. Thus, $n = \mathrm{poly}(w)$ is the regime of interest in the context of general space-bounded derandomization (see, e.g., [Nis92, Nis94, INW94]). Nonetheless, studying the problem for arbitrary n, w has attracted substantial attention in the literature and proved useful for obtaining important results in the $n = \mathrm{poly}(w)$ case. We turn to give a brief historic account on both regimes.

The $n \ll w$ regime. Nisan and Zuckerman [NZ96] proved that any space-$\log w$ randomized algorithm that uses $n = \mathrm{poly}(\log w)$ coins can be simulated deterministically in space $O(\log w)$. Other works include [Arm98] who considered a different range of parameters, and the work of Raz and Reingold [RR99] that put forth an approach for significantly improving upon [NZ96]. Most relevant to us is the work of Saks and Zhou [SZ99] which builds on Nisan's work [Nis92]. Interestingly, a recent line of work [PV21, HPV21, PV22, BHPP22] studies restricted models for *unbounded* w.

The $n \gg w$ regime. The other extreme case has been extensively studied in the *black-box* model by analyzing the structure of the corresponding non-uniform model of read once branching programs [RSV13, SVW17, FK18]. In particular, Meka, Reingold and Tal [MRT19] constructed a PRG against width $w = 3$ branching programs with seed length $\widetilde{O}(\log n)$. There has been exciting line of work on more restricted models in this regime (see, e.g., [BRRY14, KNP11, De11, Ste12, DMR$^+$21] and references therein).

The $n \gg w$ regime: the white-box model. The focus of this work is the latter regime, $n \gg w$, in the *white-box* model. I.e., instead of trying to construct a PRG against width-

w length-n branching programs, our goal is to approximate A^n for a stochastic $w \times w$ matrix A in bounded space. Even more ambitiously, we wish to handle the Iterated Matrix Multiplication (IMM) problem for stochastic matrices, that is, to approximate the product $A_1 \cdots A_n$ for arbitrary stochastic matrices. We turn to briefly survey the known results.

Savitch's theorem [Sav70] can be adapted to obtain an exact computation of the product of arbitrary matrices in space $O(\log n \cdot \log(nw))$ (ignoring bit representation issues, see Claim 3.6). Allowing for an approximation error $\varepsilon > 0$, one can implement Savitch's algorithm using standard techniques in $O(\log n \cdot (\log w + \log\log \frac{n}{\varepsilon}))$ space. For *stochastic* matrices the seminal Saks–Zhou algorithm [SZ99] runs in space

$$O\left(\sqrt{\log n} \cdot \log \frac{nw}{\varepsilon}\right).[1]$$

The dependence on ε was recently improved by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford, and Vadhan [AKM+20] using the Richardson iteration (see Section 3.4), reducing the space to

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

Hoza [Hoz21] gave a poly-logarithmic improvement in the $n = \mathrm{poly}(w)$ regime, attaining $O(\frac{1}{\sqrt{\log\log n}} \cdot \log^{3/2} n)$ space, also in the small error regime. We refer the reader to the excellent, very recent survey by Hoza [Hoz22] on the progress on derandomizing space-bounded computation.

1.1 Our result

The main result of this work is as follows.

Theorem 1.1 (see also Theorem 6.1)**.** *For any $n, w \in \mathbb{N}$ where $n \geq w$, and any $\varepsilon > 0$, there exists a deterministic algorithm that given $w \times w$ stochastic matrices $A_1, \ldots, A_n$, approximates $A_1 \cdots A_n$ to within error $\varepsilon = 2^{-\mathrm{polylog}(n)}$ in space*

$$\widetilde{O}\left(\log n + \sqrt{\log n} \cdot \log w\right),$$

where the $\widetilde{O}$ notation hides doubly-logarithmic factors in n and w.

Theorem 1.1 gives the first *polynomial* improvement over [SZ99] (and over [AKM+20])

[1][SZ99] considers matrix powering. However, one can reduce IMM to matrix powering via the embedding $(A_1, \ldots, A_n) \mapsto \bar{A} = \begin{pmatrix} 0 & & & \\ A_1 & 0 & & \\ & \ddots & \ddots & \\ & & A_n & 0 \end{pmatrix}$. Indeed, $A_1 \cdots A_n$ appears as an entry in $\bar{A}^n$. We note that this simple reduction, that incurs a "blow up" $w \mapsto nw$, is moot in our regime of interest, $n \gg w$.

for IMM and even for matrix powering. In particular, in the regime $\log n > \log^2 w$, our algorithms runs in $\widetilde{O}(\log n)$ space compared to the previous best $O(\log^{3/2} n)$.

1.2 The case of matrix powering

For the case of matrix powering, we osberve that an exact algorithm that is based on the Cayley–Hamilton theorem yields the following.

Theorem 1.2. *For any $n, w \in \mathbb{N}$ there exists a deterministic algorithm that on input a $w \times w$ matrix A, represented by $\mathrm{poly}(w)$ bits, outputs A^n using space $O(\log n + \log^2 w)$.*

Although the proof of Theorem 1.2 uses standard linear algebra and known results from parallel circuit complexity, we are not aware of any reference in which it is explicitly stated. For completeness, we give the proof in Appendix A. Our algorithm given by Theorem 1.1 outperforms previous matrix powering algorithms, including Theorem 1.2, whenever $\log w \ll \log n \ll \log^2 w$. We stress that we are not aware of any algorithm, spectral or otherwise, attaining such a space complexity for IMM as in Theorem 1.2.

There are two natural ways to further interpret our result for matrix powering.

Approximating long random walks. Our result yields approximation of long random walks on arbitrary digraphs with super-polynomial mixing time. Letting A be a $w \times w$ stochastic matrix, $n = n(w) \gg w$, and $v \in \mathbb{R}^w$ be any initial distribution, Theorem 1.1 gives a space-efficient algorithm for approximating $A^n v$, outperforming previous methods. When A corresponds to an irreducible and aperiodic Markov chain with a *polynomial* mixing time, $n = \mathrm{poly}(w)$ already suffices for $A^n v$ to be very close to the stationary distribution. When the underlying Markov chain is not poly-mixing, which is often the case for arbitrary digraphs, the regime $n \gg w$ may give us valuable information.

Space-bounded derandomization. In the lens of derandomization, Theorem 1.1 proves that any randomized algorithm that uses n random bits and S space can be simulated deterministically in $O(\log n + \sqrt{\log n} \cdot S)$ space. In the regime $n = 2^{S^c}$ for $c > 1$, where our algorithm shines, there is a subtlety that one should bear in mind. Conventionally, randomized algorithms use at most $2^{O(S)}$ random coins. Otherwise, the algorithm reaches the same state twice, implying that there are (infinite) sequences of random coins for which the algorithm never terminates. To settle the halting issue, it is natural to consider the model in which a randomized algorithm uses S space, and n random coins *in expectation*. With this modification, our simulation result holds. We make two additional remarks: (1) For $n = 2^{O(S)}$, the above modification agrees with the standard model; and (2) Taking

$n \gg 2^S$ may decide languages outside **BPL**, e.g., if $n = 2^{2^{O(S)}}$ then we can decide the directed connectivity problem which is not known to be in **BPL**.

Interestingly, as noted by Hoza [Hoz22], the early works on randomized space-bounded algorithms showed more interest in the "non-halting" model (see, e.g., [Gil77, Sim81, Jun81, BCP83, Sak96]).

2 Proof Overview

In this section we give a high-level, yet comprehensive, overview of the proof of Theorem 1.1. The proof involves several new ideas, many of which appear already in the special case of matrix powering. There, we harness recent machinery from time- and space-bounded Laplacian solvers to the [SZ99] framework and devise an intricate precision-alternating recursive scheme. We elaborate on this in Section 2.1. The general case of iterated matrix multiplication poses additional significant challenges, the substantial of which is handled by devising an improved shift and truncate mechanism. The new mechanism is made possible by a novel use of the Richardson iteration. We present the main ideas that go into the IMM algorithm in Section 2.2.

2.1 Matrix powering

2.1.1 The [SZ99] algorithm: a refresher

Our result is based on the beautiful Saks–Zhou algorithm which we now briefly recall. For a more complete exposition, see Section 4. The algorithm consists of two ingredients:

1. The celebrated Nisan generator [Nis92], which is used as a randomized matrix exponentiation algorithm; and

2. A *canonicalization* step that is based on the *shift and truncate* technique. By the latter, we mean subtracting a small quantity from intermediate calculations (i.e., shift), and keeping only some of the most significant bits (i.e., truncate).

Roughly speaking the [SZ99] algorithm works as follows, where, for simplicity we first consider the case $w = n$. The algorithm gets as input a stochastic matrix $A \in \mathbb{R}^{n \times n}$, auxiliary randomness for the Nisan generator as well as for the shifts, and proceeds as follows.

1. Set $\widetilde{A}_0 = A$.

2. For $i = 1, \ldots, \sqrt{\log n}$,

   (a) Invoke the Nisan generator to approximate $(\widetilde{A}_{i-1})^{2^{\sqrt{\log n}}}$ to withing accuracy acc_1.

   (b) Shift $(\widetilde{A}_{i-1})^{2^{\sqrt{\log n}}}$ by a random shift of magnitude $\mathrm{Z} \cdot \mathrm{acc}_1$, where Z is chosen uniformly at random from $\{0, 1, \ldots, L\}$ and truncate it to a precision of acc_2 to obtain the matrix $\widetilde{A}_i$.

3. Output $\widetilde{A}_{\sqrt{\log n}}$.

Setting of parameters. The parameters $L, \frac{1}{\mathrm{acc}_1}, \frac{1}{\mathrm{acc}_2}$ are all set to be sufficiently large polynomials in n that further satisfying certain relations. While the exact setting is not important for our current discussion, the reader may take $L = n^a$, $\mathrm{acc}_1 = n^{-4a}$, and $\mathrm{acc}_2 = n^{-2a}$ for some sufficiently large constant a. The reason why acc_1 and acc_2 have to be polynomially small in n is because errors accumulate additively, and if we raise to a power of n, the final error is of order $n(\mathrm{acc}_1 + \mathrm{acc}_2)$. The reason why L has to be polynomially large is because we take the union bound over all n^2 entries, and over the $\sqrt{\log n}$ iterations, resulting in failure probability $\approx \frac{n^2}{L}$.

Analyzing the space complexity. The above algorithm is randomized. However, as usual, to obtain a deterministic algorithm one can average over the choices of the auxiliary randomness which can be done in additional space that is proportional to the randomness complexity.

Let us sketch the analysis of the [SZ99] algorithm's space complexity. The crucial point in the randomized algorithm above is that the canonicalization step (which is implicit in (a) and discussed in Section 3.5) allows [SZ99] to *reuse* the randomness needed for the different applications of the Nisan generator. This reuse of randomness saves on space in the resulted deterministic algorithm. One, and hence all, application of Nisan's generator with the above parameters, requires $O(\log^{3/2} n)$ random bits. Adding to that the $O(\log n)$ random bits per shift, which we do not reuse, we get randomness complexity of $O(\log^{3/2} n)$. The space complexity of every iteration can be shown to be $O(\log n)$, yielding an overall space complexity of $O(\log^{3/2} n)$ for the randomized algorithm. Hence, the overall space complexity of the deterministic algorithm is also $O(\log^{3/2} n)$.

2.1.2 Attempting to gain on $w \ll n$ in Saks–Zhou

We turn to check what changes in the regime $w \ll n$. First, let us employ the same approach as before, i.e., we have $\sqrt{\log n}$ iterations, each raising the previously computed matrix to a power of $2^{\sqrt{\log n}}$. Then,

- As before, the parameters $\mathrm{acc}_1, \mathrm{acc}_2$ have to be $n^{-\Theta(1)}$ because the errors accumulate additively in n regardless of the matrix's dimension w.
- However, we can now take L to be smaller as there are only w^2 entries in the matrix, and we only need to take the union bound over these entires and over the $\sqrt{\log n}$ iterations, which is negligible. Indeed, our algorithm invests roughly $\log w$ random bits for choosing the shifts.[2]

Doing the back-of-the-envelope calculation of the overall space complexity, we see that we gained nothing. Indeed,

- Both the space and randomness complexity of the Nisan generator is still $\Omega(\sqrt{\log n} \cdot \log \frac{1}{\mathrm{acc}_1}) = \Omega(\log^{3/2} n)$, because acc_1 is polynomially-small in n; and,
- Each of the $\sqrt{\log n}$ iterations still requires $\Omega(\log n)$ space, because the canonicalization step has to work with accuracy of $n^{-\Theta(1)}$.

Thus, the [SZ99] algorithm does not benefit, as is, from the smaller input matrix it is given. The crux of the problem lies in the fact that we have to work with accuracy of $n^{-\Theta(1)}$ and then it seems inevitable that each of the $\sqrt{\log n}$ iterations should take $\Omega(\log n)$ space. In an amortized sense, the space complexity that we are shooting for restricts us to $\sqrt{\log n} + \log w \ll \log n$ space per iteration which seems insufficient if we are to maintain accuracy of $n^{-\Theta(1)}$.

Despite the seemingly impossible "space vs. accuracy" requirement, the novelty of our solution allows us to accomplish just that, namely, maintaining accuracy of $n^{-\Theta(1)}$ throughout the computation, at a cost of $\sqrt{\log n} + \log w \ll \log n$ bits per iteration! To explain how this is done, we pause our description of the modified Saks–Zhou algorithm to discuss Richardson iteration.

2.1.3 Richardson Iteration

Primarily used as an iterative method for solving linear systems, the Richardson iteration has been extremely useful in graph algorithms, and was recently applied in the space-bounded setting. In this work, we use it to obtain a high precision approximation of matrix powers from mild approximations, as was done in [AKM+20, PV21, CDR+21]. We turn to describe this algorithm.

[2]Note, however, that each shift is of magnitude at most $L \cdot \mathrm{acc}_1 = n^{-\Theta(1)}$.

The algorithm R gets as input a substochastic matrix A of dimension w, an integer k, and a sequence of $w \times w$ matrices $\widetilde{A}_1, \ldots, \widetilde{A}_n$ satisfying $\|A^i - \widetilde{A}_i\|_\infty \leq \frac{1}{n}$. The output, R, is a $w \times w$ matrix satisfying $\|A^n - R\|_\infty \leq n \cdot 2^{-k}$, computed in space $O\big(\log^2 k + \log k \cdot \log nw\big)$. Thus, the algorithm R allows us to obtain any desired approximation ε to A^n given only a mild, $\frac{1}{n}$, approximation of the powers $A^2, \ldots, A^n$. The algorithm does so with extremely small space. Indeed, the dependence on ε is only polynomial in $\log\log\frac{1}{\varepsilon}$. We think of the matrix A as an "anchor" – an error-free object that, information theoretically, stores all that is needed to compute A^n. With access to A, the algorithm R is able, in a space-efficient manner, to improve a modest approximation of A's powers. We refer the reader to Section 3.4 for a more complete and formal discussion.

2.1.4 Turning back to our matrix powering algorithm

We employ the following approximation scheme: Throughout the computation our matrices $\widetilde{A}_i$ are kept with $n^{-\Theta(1)}$ accuracy. However, before we apply the canonicalization step, and the Nisan generator that follows, we purposely *decrease* the precision of the input matrix to the Nisan generator by truncating its entries to a precision of $w^{-\Theta(1)} \gg n^{-\Theta(1)}$. Indeed, with this modest precision, the Nisan generator requires space of order $\sqrt{\log n} \cdot \log w \ll \log^{3/2} n$. The output of the generator then gives us a "mild" approximation of the $2^{\sqrt{\log n}}$-th power. To restore the (required) high precision approximation of $n^{-\Theta(1)}$, we invoke the *Richardson iteration* which can be done space-efficiently.

It is crucial to note that although we decrease the precision before using canonicalization and the Nisan generator to save on space, this precision is not lost because we "keep" the untruncated matrix as an anchor for the correct result: The Richardson iteration combines the untruncated matrix with the mild approximation of its $2^{\sqrt{\log n}}$-th power, to get a high-precision approximation of that power.

We are now ready to give a rough outline of our matrix powering algorithm (see also Figure 1). The precise description is given in Section 5. Our algorithm gets as input a stochastic matrix $A \in \mathbb{R}^{w \times w}$, auxiliary randomness for the Nisan generator as well as for the shifts, and proceeds as follows.

1. Set $\widetilde{A}_0 = A$.
2. For $i = 1, \ldots, \sqrt{\log n}$,
   (a) Truncate $\widetilde{A}_{i-1}$ to a precision of $w^{-\Theta(1)}$ and denote the result by $\lfloor \widetilde{A}_{i-1} \rfloor$.
   (b) Set the Nisan generator to work with accuracy $w^{-\Theta(1)}$ and use it to approximate $\lfloor \widetilde{A}_{i-1} \rfloor^{2^{\sqrt{\log n}}}$. Note that since $\widetilde{A}_{i-1}$ approximates $\lfloor \widetilde{A}_{i-1} \rfloor$ to within accuracy of

$w^{-\Theta(1)}$, we have that $\widetilde{A}_{i-1}^{2^{\sqrt{\log n}}}$ approximates $\lfloor \widetilde{A}_{i-1} \rfloor^{2^{\sqrt{\log n}}}$ to within accuracy of $w^{-\Theta(1)} \cdot 2^{\sqrt{\log n}}$.

(c) Use the mild approximation obtained above to compute a high precision approximation $R_i \approx \widetilde{A}_{i-1}^{2^{\sqrt{\log n}}}$ by applying the Richardson iteration. We stress that the Richardson iteration improves our approximation with respect to the previous high precision approximation $\widetilde{A}_{i-1}$ and not its truncation.

(d) Shift R_i by a random shift of magnitude $n^{-\Theta(1)}$, and truncate it to a precision of $n^{-\Theta(1)}$, to obtain the matrix $\widetilde{A}_i$.

3. Output $\widetilde{A}_{\sqrt{\log n}}$.

Figure 1 illustrates the alternating nature of the algorithm, zig-zagging between a mild approximation of $w^{-\Theta(1)}$ and a high precision approximation of $n^{-\Theta(1)}$. Setting the parameters appropriately, we get that with high probability over the auxiliary randomness, i.e., the seed for the Nisan generator and the shifts, the algorithm outputs a good approximation for A^n using space $\widetilde{O}(\log n + \sqrt{\log n} \cdot \log w)$.

Averaging over the auxiliary randomness, as done in [SZ99], would yield a space-efficient deterministic algorithm, albeit with accuracy of $w^{-\Theta(1)}$. It is thus tempting to try and apply an additional layer of the Richardson iteration in order to improve the accuracy to an arbitrary $\varepsilon > 0$ (as done in [AKM+20] for the standard Saks–Zhou algorithm). However, to apply the Richardson iteration, the initial accuracy needs to be $\frac{1}{n}$. To overcome this issue, we observe that while the average does not give us a good enough guarantee, the *median* does. Applying the Richardson iteration after taking the median over the auxiliary randomness, we get our final high-precision approximation.

2.2 Iterated matrix multiplication

Let us try to naïvely extend our powering algorithm, discussed in the previous section, to compute the product $A_1 A_2 \cdots A_n$ of arbitrary $w \times w$ stochastic matrices. Given $A_1, \dots, A_n$, we would proceed as follows:

1. Use Nisan generator to approximate iterated products of $2^{\sqrt{\log n}}$ matrices, instead of the $2^{\sqrt{\log n}}$-th power of a single matrix.

2. Recursively, partition the iterated product to iterated products of $2^{\sqrt{\log n}}$ matrices. After $\sqrt{\log n}$ iterations, the entire iterated product is approximated.

There are three major issues with this naïve attempt:

Working with one shift. When we needed to handle matrix powering, we invested only $O(\log w)$ random bits per shift, and we had $\sqrt{\log n}$ such shifts, one for every matrix we encounter in the computation (namely, the approximations for the matrices $A^{2^{i\cdot\sqrt{\log n}}}$ for $i = 1, \ldots, \sqrt{\log n}$). However, now we have $\Omega(n)$ intermediate matrices, and we cannot afford to use an independent shift for each nor to incur the union-bound over all $\Omega(n)$ sub-sequences.

We therefore put forth a new approach which, in a way, is the most economical approach we can think of. Instead of shifting "output" matrices (those that arise as intermediate computation, after applying Nisan's generator), we shift the *input matrices*. Also, as we have n input matrices $A_1, \ldots, A_n$, we use the *same* shift Z on all n input matrices. We need each of the shifts to work well, so we need to union-bound over n matrices, and therefore use $\Omega(\log n)$ bits for choosing the shift Z. Thus, we cannot afford to do such a shift at each iteration, and instead we study what happens when we just shift the input without shifting intermediate iterations.

While this saves space (now we can afford $O(\log n)$ random bits for shifting the input since we only worry about one iteration–the first one–rather than $\sqrt{\log n}$ of them). Analyzing correctness becomes highly nontrivial since we need to keep track of the way the matrices (as well as the error) evolve throughout the intermediate computations. We first show that the single initial perturbation makes all *original* iterated products "safe", in the sense that it does not introduce undesired dependencies. However, the truncation step makes the *approximated* matrices unsafe. Surprisingly, this is resolved by introducing a second Richardson iteration, now for the purpose of handling dependencies rather than for improving the accuracy. See Section 6.1.2 for a more detailed discussion.

Better space complexity analysis. In the space complexity analysis of the matrix powering algorithm, we use standard composition of space bounded algorithms, where the space complexity of each iteration is roughly $\Theta(\log w)$. However, in IMM there are roughly n terms in the product, and so the space complexity of each iteration is $\Omega(\log n)$, even just for keeping a fresh index to a multiplication interval at each level of the composition. Thus, seemingly, the total space complexity is $\Omega(\log^{3/2} n)$. We resolve this issue by observing that some of the indices can be maintained *globally*. See Section 6.5.1 for the details, and in particular Lemma 6.13 that generalizes the standard space composition theorem.

The confidence parameter. In the matrix powering algorithm, Nisan generator has to work against all matrices $A^{2^{i\cdot\sqrt{\log n}}}$ for $i = 0, 1, \ldots, \sqrt{\log n} - 1$. As there are only $\sqrt{\log n}$ matrices to consider, we could choose a large confidence parameter $\delta \approx \frac{1}{w}$ (see, e.g., Section 3.5 or Theorem 3.14) and still be certain that with probability $1 - \delta$ over the auxiliary

$$\begin{tikzcd}
\widetilde{M_0} \arrow[r, "\text{Richardson}"] \arrow[d, "\text{Truncate}"'] & \mathsf{R}_0 \arrow[r, "\text{S\&T}"'] & \widetilde{M_1} \arrow[r, "\text{Richardson}"] \arrow[d, "\text{Truncate}"'] & \mathsf{R}_1 \arrow[r, "\text{S\&T}"'] & \widetilde{M_2} \\
\lfloor \widetilde{M_0} \rfloor \arrow[r, "\text{Nisan}"'] & \approx \lfloor \widetilde{M_0} \rfloor^{2^{\sqrt{\log n}}} \arrow[u, "\text{Richardson}"] & \lfloor \widetilde{M_1} \rfloor \arrow[r, "\text{Nisan}"'] & \approx \lfloor \widetilde{M_1} \rfloor^{2^{\sqrt{\log n}}} \arrow[u, "\text{Richardson}"] & \cdots
\end{tikzcd}$$

Figure 1: Our matrix powering algorithm. "S & T" refers to "shift and truncate".

randomness for the Nisan generator h, our choice works well for all $\sqrt{\log n} \ll w$ matrices above.

In contrast, for the IMM algorithm, we need to fix a single h that works well against each of the $\Omega(n)$ sub-products. Therefore, the confidence deteriorates to $\approx n \cdot \delta$ which forces us to take $\delta < \frac{1}{n}$. However, in this parameter setting the Nisan generator has seed length $\Omega(\log^{3/2} n)$ which is too much for us. We remedy this by devising a PRG with a better dependence on the confidence parameter δ. This is done by standard techniques (see Section 6.1.1 for more details).

3 Preliminaries

3.1 Matrix notation

For a matrix $A \in \mathbb{R}^{w \times w}$, we denote $\|A\|_{\max} = \max_{i,j \in [w]} |A[i,j]|$ and by $\|A\|_\infty$ we denote its induced ℓ_∞ norm, i.e., $\|A\|_\infty = \max_{i \in [w]} \sum_{j \in [w]} |A[i,j]|$. Clearly,

Claim 3.1. *For any matrix $M \in \mathbb{R}^{w \times w}$ we have that $\|M\|_\infty \leq w \|M\|_{\max}$.*

We say a real matrix is *stochastic* if it is row-stochastic, i.e., if its entries are nonnegative and every row sums to 1. We say that a real matrix is *substochastic* if its entries are non-negative and every row sums to at most 1, i.e., $\|A\|_\infty \leq 1$. The following claim follows by a simple induction and the triangle inequality.

Claim 3.2. *Let $\|\cdot\|$ be a sub-multiplicative matrix norm. Then, for any $A_1, \ldots, A_k, B_1, \ldots, B_k$*

with norm at most 1 *we have that*

$$\|A_1 \cdots A_k - B_1 \cdots B_k\| \leq \sum_i \|A_i - B_i\| .$$

In particular, if $\|A\|, \|B\| \leq 1$ *then* $\left\|A^k - B^k\right\| \leq k \cdot \|A - B\|$.

3.2 Space-bounded computation

A deterministic space-bounded Turing machine has three tapes: an input tape (that is read-only); a work tape (that is read-write) and an output tape (that is write-only and uni-directional). The output of the TM is the content of its output tape once the machine terminates. The space used by a TM M on input x is the rightmost work tape cell that M visits upon its execution on x. Denoting this quantity by $s_M(x)$, the space complexity of M is thus the function $s(n) = \max_{x:|x|=n} s_M(x)$. For further details, see [AB09, Chapter 4] and [Gol08, Chapter 5].

Claim 3.3 (composition of space-bounded algorithms). *Let* $f_1, f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$ *be functions that are computable in space* $s_1, s_2 \colon \mathbb{N} \to \mathbb{N}$*, respectively, where* $s_1(n), s_2(n) \geq \log n$*. Then,* $f_1 \circ f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$ *can be computed in space* $O\left(s_1\left(\ell_2(n)\right) + s_2(n)\right)$*, where* $\ell_2(n)$ *is a bound on the output length of* f_2 *on inputs of length* n*.*

Corollary 3.4. *Let* $f \colon \{0,1\}^\star \to \{0,1\}^\star$ *be computable in space* $s \colon \mathbb{N} \to \mathbb{N}$*, where* $s(n) \geq \log n$*. Then,* $g(x,k) = f^{(k)}(x)$*, where* $k \in \mathbb{N}$*, can be computed in space*

$$O\left(\sum_{i=1}^{k-1} s\left(\ell_i(n)\right)\right)$$

where $\ell_i(n)$ *is a bound on the output length of* $f^{(i)}$ *on inputs of length* n*.*

Next, we recall the space complexity of computing matrix powers via naïve repeated squaring. Observe that whenever two numbers are multiplied, their multiplication requires more digits of precision and so we have to account for that as well.

Definition 3.5 (matrix bit complexity). *Given a matrix* $A \in \mathbb{R}^{w \times w}$*, we denote its bit complexity, i.e., the number of bits required to represent all its entries, by* $|A|$*. In particular, if we use* k *bits of precision for every entry in* A *then* $|A| = O(kw^2)$*. We will always assume* $|A| = \Omega(w^2)$*.*

Claim 3.6. *The matrix powering function* $f(A, n) = A^n$ *can be computed in space* $O(\log^2 n + \log n \cdot \log |A|)$.

Proof. First, note that the product of two matrices $f(A,B) = AB$ can be computed in space $O(\log(|A| + |B|))$, where we use our assumption $|A|, |B| = \Omega(w^2)$. Composing $f(A,A)$ for k times, we can compute A^{2^k} in space

$$O\left(\sum_{i=1}^{k} \log\left(2^i |A|\right)\right) = O\left(k^2 + k \log |A|\right),$$

following Corollary 3.4. (Note that the number of bits needed to represent each entry doubles at every iteration). Write $n = \sum_{i=0}^{\lceil \log n \rceil} b_i 2^i$ for $b_i \in \{0,1\}$. Then, $A^n = \prod_{i:b_i=1} A^{2^i}$. Accounting for the $\log n$ additional space needed to compute the product, the proof is concluded. □

3.3 Read-once branching programs

We use the standard definition of layered read-once branching programs. For a length parameter $n \in \mathbb{N}$, a width parameter $w \in \mathbb{N}$, and an alphabet Σ, an $[n, w, \Sigma]$ BP is specified by an initial state $v_0 \in [w]$, a set of accept states $V_{\mathsf{acc}} \subseteq [w]$ and a sequence of transition functions $B_i \colon [w] \times \Sigma \to [w]$ for $i \in [n]$. The BP B naturally defines a function $B \colon \Sigma^n \to \{0,1\}$: Start at v_0, and then for $i = 1, \ldots, n$ read the input symbol x_i and transition to the state $v_i = B_i(v_{i-1}, x_i)$. The BP accepts x, i.e., $B(x) = 1$, if $v_n \in V_{\mathsf{acc}}$, and rejects otherwise.

Given a transition function B_i, and $\sigma \in \Sigma$, we identify the function $B_i(\cdot, \sigma) \colon [w] \to [w]$ with a Boolean stochastic matrix which we denote $B_i(\sigma)$, wherein $B_i(\sigma)[u,v] = 1$ if and only if $B_i(u, \sigma) = v$. The transition matrix of each layer corresponds to the matrix $\mathsf{A}(B_i) \triangleq \mathbb{E}_{\sigma \in \Sigma}\left[B_i(\sigma)\right]$. The transition matrix of B itself is thus

$$\mathsf{A}(B) \triangleq \mathsf{A}(B_1) \cdot \ldots \cdot \mathsf{A}(B_n),$$

which describes a uniformly random walk on B starting at v_0. In particular, the probability that B accepts a random input is given by $\sum_{v \in V_{\mathsf{acc}}} \mathsf{A}(B)[v_0, v]$. In our work we will approximate $\mathsf{A}(B)$ in a strong sense that would be oblivious to the initial state and the set of accepting states, so we will never mention them explicitly. Namely, if M is such that $\|\mathsf{A}(B) - M\|_\infty \le \varepsilon$, we ε-approximate the aforementioned acceptance probability for any v_0 and V_{acc}.

Finally, when we omit the length of the BP and simply refer to B as a $[w, \Sigma]$ BP, we mean that B comprises a single transition function, and we sometimes repeat it for, say, n times, to mimic the length-n BP in which every transition is the same as this of B. This notion is very natural, and in fact suffices, when one wishes to approximate *powers* of stochastic matrices rather than iterated matrix multiplication. Given a $[w, \Sigma]$ BP B with

$\mathsf{A}(B) = A$, A^n is thus the transition matrix of the BP with n identical transitions.

3.4 Richardson Iteration

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, and has recently been used in the context of space-bounded computation in [AKM+20, PV21, CDR+21]. We describe it formally.

Definition 3.7 (Richardson iteration). *Given $A, B \in \mathbb{R}^{w\times w}$, and $k \in \mathbb{N}$, we define*

$$\mathsf{R}(A, B, k) = \sum_{i=0}^{k-1} (I - AB)^i A.$$

Above, one can think of B as the Laplacian of some stochastic matrix, and of A as a coarse approximation of its inverse.

Lemma 3.8. *For any sub-multiplicative norm $\|\cdot\|$, let $A, B \in \mathbb{R}^{w\times w}$ be such that $\|I - AB\| \le \varepsilon$ and B is invertible. Then, $\|\mathsf{R}(A, B, k) - B^{-1}\| \le \|B^{-1}\| \cdot \varepsilon^k$.*

The above lemma can be used to devise an algorithm that improves the accuracy of matrix powers [AKM+20, PV21, CDR+21], as we state below. For completeness, we provide the short proof in Appendix B.

Lemma 3.9. *There exists an algorithm R that gets as input a sequence of substochastic matrices $(A_1, \dots, A_n)$ of dimension $w \times w$, an integer $k \in \mathbb{N}$, and a sequence substochastic matrices $(B_{i,j})_{1\le i<j\le n}$ satisfying:*

- *If for all $1 \le i < j \le n$ we have that $\|A_i \cdots A_j - (B)_{i,j}\|_\infty \le \frac{1}{4(n+1)}$, then*

$$\|\mathsf{R}((B_{i,j})_{1\le i<j\le n}, (A_i)_{1\le i\le n}, k) - A_1 \cdots A_n\|_\infty \le (n+1) \cdot 2^{-k}.$$

- *R runs in $O\big(\log^2 k + \log k \cdot \log(nT)\big)$ space, where $T = \max\{|A_i|, |(B)_{i,j}|\}$ is the maximum bit-complexity of the given matrices.*

In the above lemma, whenever $A_1 = A_2 = \ldots = A_n$ then it suffices to get as input matrices $(B_1, \dots, B_n)$ satisfying

$$\left\|A^i - B_i\right\|_\infty \le \frac{1}{4(n+1)}.$$

In this case, we shall invoke the algorithm using $\mathsf{R}(B_1, \dots, B_n, A, k)$, where $A = A_1 = A_2 = \ldots = A_n$.

3.5 The Nisan Generator

Nisan, in his seminal work [Nis92], constructed a family of pseudorandom generators that ε-fool $[n, w, \Gamma]$ BPs using seed of length $d = O\left(\log n \cdot \log \frac{nw|\Gamma|}{\varepsilon}\right)$. We briefly recall the construction and its properties.

Set the generator's "working alphabet" Σ, where $|\Sigma| = O\left(\frac{nw|\Gamma|}{\varepsilon}\right)$,[3] and let $\mathcal{H} \subseteq \Sigma \to \Sigma$ with $|\mathcal{H}| = |\Sigma|^2$ be a two-universal family of hash functions. The seed for

$$G = G_{\log n} \colon \{0,1\}^d \to \Gamma^n$$

comprises $\log n$ hash functions $h = (h_1, \ldots, h_{\log n})$, each $h_i \in \mathcal{H}$, and a symbol $\sigma \in \Sigma$, noticing that indeed $d = O(\log n \cdot \log |\Sigma|)$. We define

$$G_i \colon \Sigma \times \{0,1\}^{i \cdot 2 \log |\Sigma|} \to \Gamma^{2^i}$$

recursively as follows.

$$\begin{aligned} G_0(\sigma) &= \sigma|_{[1,\ldots,\log|\Gamma|]}, \\ G_i(\sigma; h_1, \ldots, h_i) &= G_{i-1}(\sigma; h_1, \ldots, h_{i-1}) \circ G_{i-1}(h_i(\sigma); h_1, \ldots, h_{i-1}). \end{aligned}$$

One can verify that the space needed to compute the output of G, given an appropriate $\mathcal{H}$, is $O(\log |\Sigma|) \ll d$. Nisan proved that for every BP B, most $h = (h_1, \ldots, h_{\log n})$ are good in the sense that $G(\Sigma, h)$ ε-fools B. Formally,

Theorem 3.10 ([Nis92]). *Given $n, w \in \mathbb{N}$, an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and an alphabet Γ, let $G \colon \{0,1\}^{d_{\mathsf{N}}} \times \Sigma \to \Sigma^n$ be the above Nisan generator, where $|\Sigma| = O\left(\frac{nw|\Gamma|}{\varepsilon\delta}\right)$ and $d_{\mathsf{N}} = O\left(\log n \cdot \log |\Sigma|\right)$. Let B be any $[n, w, \Gamma]$ BP. Then, with probability at least $1 - \delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$, it holds that*

$$\left\| \mathsf{A}(B) - \mathop{\mathbb{E}}_{\sigma \in \Sigma}\left[B(G(h, \sigma))\right] \right\|_\infty \leq \varepsilon,$$

recalling that $\mathsf{A}(B) = \mathbb{E}_{z \in \Gamma^n}\left[B(z)\right]$.[4]

For every BP B, if we choose h at random and store it then h is good for B with probability $1 - \delta$. Thus, we can write h once and never change it. Put differently, the

[3] If Γ is large enough already, we can simply take $\Sigma = \Gamma$, but the above choice of Σ will not change the parameters.

[4] We note that one can view the output of Nisan generator as a $[w, \Sigma]$ BP $B_h^{(n)}$ whose transition matrix $\mathsf{A}(B_h^{(n)})$ is precisely $\mathbb{E}_{\sigma \in \Sigma}\left[B(G(h, \sigma))\right]$.

storage needed to keep h is a *write-once* memory. In contrast, the storage needed to keep σ is a multiple-read, multiple-write memory, as we need to average over σ. It turns out that this distinction is incredibly useful. Saks and Zhou call the *write-once* storage "offline" randmoness, and the *multiple-write* storage "online" randomness.

Canonicalization of BPs. As discussed above, a BP B has an associated transition matrix $A(B)$. This association, however, is not one to one, and there are many different BPs that share the same associated transition matrix A. An important step in [SZ99] is to transform a BP B to a canonical BP B' that has the same associated transition matrix. We first make this notion formal.

Given a $w \times w$ substochastic matrix M in which every entry is represented using at most s bits, let $B = \mathsf{C}(M)$ be the $[w+1, \Sigma = [2^s]]$ BP constructed as follows. Given $i \in [w]$ and $\sigma \in \Sigma$, $B(i, \sigma) = j$ where j is the smallest integer satisfying $\sum_{k \le j} M[i,k] \ge \sigma \cdot 2^{-s}$ if such exists, and $w+1$ otherwise. Moreover, we set $B(w+1, \sigma) = w+1$ for all $\sigma \in \Sigma$. The following claim then follows easily.

Claim 3.11. *For a substochastic matrix M, it holds that $\mathsf{A}(\mathsf{C}(M))_{[1,w]} = M$, where we denote by $A_{[a,b]}$ the sub-matrix of A that is formed by taking the rows and columns indexed by $a, \dots, b$.*

In our work, we will also need to work with *lossy* canonicalizations, in which we translate a substochastic matrix with a large bit-complexity into a BP over a small alphabet. Given a substochastic M and $t \in \mathbb{N}$, we let $\mathsf{C}_t(M)$ be the canonicalization of M into a BP of width $w+1$ over the alphabet $\Sigma = \{0,1\}^t$, *regardless* of the representation of its elements. Namely, $B = \mathsf{C}_t(M)$ is defined such that $B(i, \sigma) = j$, where again, j is the smallest integer satisfying $\sum_{k \le j} M[i,k] \ge \sigma \cdot 2^{-t}$ if such exists, and $w+1$ otherwise. We also set $B(w+1, \sigma) = w+1$ for all $\sigma \in \Sigma$ as before.

Claim 3.12. *Let M be a $w \times w$ substochastic matrix M in which every entry is represented using at most s bits, and let $t \in \mathbb{N}$ where $t \le s$. Then,*

$$\left\| \mathsf{A}(\mathsf{C}_t(M))_{[1,w]} - M \right\|_\infty \le w \cdot 2^{-t}.$$

Moreover, computing C_t takes $O(\log s + \log w)$ space.

An Extended Nisan Algorithm. For simplicity, let us only consider a $[w, \Sigma]$ BP with a transition matrix A rather than different transitions at each layer. Observe that the Nisan generator, set with length parameter n, can also approximate all intermediate powers by truncating its output accordingly. Thus:

Theorem 3.13 (following [Nis92]). *There exists an algorithm* N *that gets as input a* $[w,\Sigma]$ *BP* B *with a transition matrix* $A=\mathsf{A}(B)$*, a length parameter* n*, an accuracy parameter* $\varepsilon>0$*, a confidence parameter* $\delta>0$*, and a seed* $h\in\{0,1\}^{d_{\mathsf{N}}}$ *where* $d_{\mathsf{N}}=O\left(\log n\cdot\log\frac{nw|\Sigma|}{\varepsilon\delta}\right)$*. The algorithm runs in space* $O\left(\log\frac{nw|\Sigma|}{\varepsilon\delta}\right)$ *and outputs*

$$\left(M_h^{(1)},\ldots,M_h^{(n)}\right)=\mathsf{N}_{\varepsilon,\delta}(B,h,n),$$

each $M_h^{(i)}\in\mathbb{R}^{w\times w}$*, and satisfies the following. With probability at least* $1-\delta$ *over* $h\in B^{d_{\mathsf{N}}}$*, it holds that for all* $i\in[n]$*,*

$$\left\|M_h^{(i)}-A^i\right\|_\infty\le\varepsilon.$$

We will often want to feed Nisan's algorithm with stochastic (or even substochastic) matrices, rather than BPs. The following theorem extends upon Theorem 3.13 by preforming a canonicalization step prior to applying Nisan's algorithm, and even allows for a lossy canonicalization step which would be useful toward reducing the space requirements. As it will be clear from context, we use N for both the algorithm that gets a BP as input and for the one that gets a matrix as input.

Theorem 3.14. *There exists an algorithm* $\mathsf{N}_{\varepsilon,\delta}$ *that gets as input:*

1. *A* $w\times w$ *substochastic matrix* A *in which every entry is represented using at most* s *bits.*
2. *An accuracy parameter* $\varepsilon>0$*, a confidence parameter* $\delta>0$*, and a canonicalization parameter* $t\in\mathbb{N}$*, where* $t\le s$*.*
3. *A seed* $h\in\{0,1\}^{d_{\mathsf{N}}}$ *for* $d_{\mathsf{N}}=O\left(\log n\cdot\left(t+\log\frac{nw}{\varepsilon\delta}\right)\right)$*.*

The algorithm runs in space $O\left(t+\log s+\log\frac{nw}{\varepsilon\delta}\right)$ *and outputs*

$$\left(M_h^{(1)},\ldots,M_h^{(n)}\right)=\mathsf{N}_{\varepsilon,\delta}(A,h,n,t),$$

each $M_h^{(i)}\in\mathbb{R}^{w\times w}$*, and satisfies the following. With probability at least* $1-\delta$ *over* $h\in\{0,1\}^{d_{\mathsf{N}}}$*, it holds that for all* $i\in[n]$*,*

$$\left\|M_h^{(i)}-A^i\right\|_\infty\le\varepsilon+nw\cdot2^{-t}.$$

When we omit the parameter t*, we implicitly set* $t=s$*, and then the error guarantee is simply* ε*. Also, when we set* N *to output a single matrix, we take it to be* $M_h^{(n)}$*.*

Proof. We compute $B=\mathsf{C}_t(A)$ and apply $\mathsf{N}(B,h,n)$, which outputs $M_h^{(1)},\ldots,M_h^{(n)}$. We then consider only the first w rows and columns of each matrix. By Theorem 3.13, with probability at least $1-\delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$, we are guaranteed that

$$\left\| M_h^{(i)} - \mathsf{A}(B)^i \right\|_\infty \le \varepsilon$$

for all $i \in [n]$. By Claim 3.12, $\|\mathsf{A}(B) - A\|_\infty \le w \cdot 2^{-t}$, and thus, due to Claim 3.2,

$$\left\| M_h^{(i)} - A^i \right\|_\infty \le \varepsilon + iw \cdot 2^{-t}.$$

The space requirements and the bound for d_{N} readily follows from Claim 3.12 and Theorem 3.13. Note that when $t = s$, the canonicalization is lossless. □

4 Background: The Saks–Zhou Algorithm

We review Saks and Zhou's algorithm, presenting it using a terminology which would allow us to lay the groundwork for our improved algorithm given in the next sections. We begin with recalling the machinery of *shift and truncate*.

4.1 Shift and Truncate

Definition 4.1 (truncation). *For $z \in \mathbb{R}$ and $t \in \mathbb{N}$, we define the truncation operator $\lfloor z \rfloor_t$ which truncates z after t bits. Namely,*

$$\lfloor z \rfloor_t = \max\left\{2^{-t} \cdot \lfloor 2^t z \rfloor, 0\right\}.$$

We extend it to matrices in an entry-wise manner. That is, for a substochastic matrix A, the matrix $\lfloor A \rfloor_t$ has entries $\lfloor A[i,j] \rfloor_t$.

Lemma 4.2. *Let $y, z \in [0,1]$ be such that $|y - z| \le 2^{-2t}$. Then, for all $\ell < t$ we have that*

$$\Pr_{\mathrm{Z}}\left[\lfloor z - \mathrm{Z} \cdot 2^{-2t} \rfloor_t \ne \lfloor y - \mathrm{Z} \cdot 2^{-2t} \rfloor_t\right] \le 2^{-\ell},$$

where Z is chosen uniformly at random from $\left\{0, 1, 2, \ldots, 2^\ell - 1\right\}$.

Proof. Without the loss of generality assume $z < y$. Note that $\lfloor z - \mathrm{Z}2^{-2t} \rfloor_t \ne \lfloor y - \mathrm{Z}2^{-2t} \rfloor_t$ is equivalent to

$$\exists a \in \mathbb{N}, \quad a2^{-t} \in \left[z - \mathrm{Z} \cdot 2^{-2t}, y - \mathrm{Z} \cdot 2^{-2t}\right). \tag{1}$$

However, by our assumption $|y-z| \le 2^{-2t}$ the following union

$$\bigcup_{\mathrm{Z} \in \{0,\dots,2^\ell-1\}} \left[z - \mathrm{Z}\cdot 2^{-2t}, y - \mathrm{Z}\cdot 2^{-2t}\right) \subseteq \left[z - (2^\ell - 1)2^{-2t}, y\right) = I$$

is disjoint and contained in the interval I which is of length at most $|y-z| + (2^\ell - 1)2^{-2t} \le 2^{-t}$. Hence, there is at most one point in I which is an integer multiple of 2^{-t}, meaning that there is at most one Z satisfying Equation (1). □

The preceding lemma is an important ingredient in [SZ99], that enables one to eliminate dependencies between consecutive applications of Nisan's algorithm. Think of z as an approximation to some y obtained by a randomized algorithm that typically returns a good approximation $z \approx y$. Note that while z, y might be extremely close, their truncation may differ if they are on the *boundary* values of the truncation operator. The idea behind Lemma 4.2 is that if we randomly shift both y, z then their truncation is equal with high probability. Once we fix a good shift, our approximation depends only on the input (and the fixed shift) and not on the internal randomness used to compute z. See [Ta-13, HK18, HU21] for additional discussion. Extending Lemma 4.2 to matrices, a simple union-bound gives us the following corollary.

Corollary 4.3. *Let $M, M' \in \mathbb{R}^{w\times w}$ be such that $\|M - M'\|_{\max} \le 2^{-2t}$. Then, for all $\ell < t$, we have that*

$$\Pr_{\mathrm{Z}}\left[\lfloor M - \mathrm{Z}\cdot 2^{-2t} J_w \rfloor_t \ne \lfloor M' - \mathrm{Z}\cdot 2^{-2t} J_w \rfloor_t\right] \le w^2 2^{-\ell},$$

where Z is chosen uniformly at random from $\left\{0, 1, 2, \dots, 2^\ell - 1\right\}$ and J_w is the all-ones $w \times w$ matrix.

4.2 The Saks–Zhou algorithm and its analysis

Given a $w \times w$ stochastic matrix A, we wish to compute A^n, where $n = 2^r$ for some integer r (n can be assumed to be a power of 2 without loss of generality). In this section we describe Saks and Zhou's randomized algorithm that uses only $O(r^{3/2})$ random bits, and runs in space $O(r^{3/2})$. As discussed toward the end of this section, the algorithm can then be derandomized in a straightforward manner while maintaining space complexity $O(r^{3/2})$.

Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ be the desired confidence. Write $r = r_1 r_2$ for some $r_1, r_2 \in \mathbb{N}$ to be chosen later on. Set $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, $t = \log \frac{2nw^2 r_2}{\varepsilon\delta}$, $\ell = \frac{t}{2}$,

$\varepsilon_{\mathsf{N}} = 2^{-2t}$, and

$$d_{\mathsf{N}} = O\left(r_1 \cdot \left(t + r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)\right) = O\left(r_1^2 + r_1 \log \frac{nw}{\varepsilon\delta}\right).$$

Without loss of generality we may assume that the input matrix A is given to us using t digits of precision. The algorithm gets as input $A \in \mathbb{R}^{w\times w}$, $r = r_1 r_2$, $h \in \{0,1\}^{d_{\mathsf{N}}}$, and $\mathrm{Z} = (\mathrm{Z}_1, \dots, \mathrm{Z}_{r_2}) \in \left\{0, \dots, 2^{\ell} - 1\right\}^{r_2}$, and proceeds as follows.

1. Set $\widetilde{M}_0 = A$.
2. For $i = 1, \dots, r_2$,
   (a) Compute $\widetilde{M}_{i-1}^{(2^{r_1})} = \mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}\left(\widetilde{M}_{i-1}, h, 2^{r_1}\right)$.
   (b) Set $\widetilde{M}_i = \left\lfloor \widetilde{M}_{i-1}^{(2^{r_1})} - \mathrm{Z}_i \cdot 2^{-2t} J_w \right\rfloor_t$.
3. Output $\widetilde{M}_{r_2}$.

Theorem 4.4 ([SZ99]). *For any $w \times w$ stochastic matrix A, and integers r_1, r_2 such that $r_1 r_2 = r = \log n$, the above algorithm satisfies the following. With probability at least $1 - \delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$ and $\mathrm{Z} = (\mathrm{Z}_1, \dots, \mathrm{Z}_{r_2}) \in \left\{0, \dots, 2^{\ell} - 1\right\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}(A, r_1, r_2, h, \mathrm{Z})$ satisfies*

$$\left\| A^n - \widetilde{M}_{r_2} \right\|_\infty \le \varepsilon.$$

Moreover, $\mathsf{SZ}(A, r_1, r_2, h, \mathrm{Z})$ runs in space $O\left(r_2 \cdot \log \frac{nw}{\varepsilon\delta}\right)$.

Proof. We let M_i be the "true" random rounding. That is, $M_0 = A$, and for each $i \in [r_2]$,

$$M_i(\mathrm{Z}) = \lfloor M_{i-1}(\mathrm{Z})^{2^{r_1}} - \mathrm{Z}_i \cdot 2^{-2t} J_w \rfloor_t. \tag{2}$$

Observe that the M_i-s do not depend on h. For brevity, we omit the dependence on Z whenever it is clear from context. It is important to note that whenever $\mathrm{Z}_1, \dots, \mathrm{Z}_i$ are fixed, the matrices $M_1, M_2, \dots, M_i$ are fixed as well, as opposed to the matrices $\widetilde{M}_1, \dots, \widetilde{M}_i$, which depend on the choice of h.

Next, we argue that with high probability (over h and the Z-s), $\widetilde{M}_i = M_i$. Toward this end, define for each fixing of $\mathrm{Z}_1, \dots, \mathrm{Z}_i$,

$$\mathrm{GOOD}_{i,\mathrm{Z}} = \left\{ h \in \{0,1\}^{d_{\mathsf{N}}} : \left\| M_i^{2^{r_1}} - \mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}\left(M_i, h, 2^{r_1}\right) \right\|_\infty \le \varepsilon_{\mathsf{N}} \right\}. \tag{3}$$

By Theorem 3.14, we get that for any $i \in [r_2]$ and $\mathrm{Z}_1, \dots, \mathrm{Z}_i$,

$$\Pr_{h \in \{0,1\}^{d_{\mathsf{N}}}} \left[h \in \mathrm{GOOD}_{i,\mathrm{Z}} \right] \geq 1 - \delta_{\mathsf{N}}. \tag{4}$$

Claim 4.5. *It holds that*

$$\Pr_{h,\mathrm{Z}} \left[\exists j \in [r_2],\ M_j \neq \widetilde{M_j} \right] \leq r_2 w^2 2^{-\ell} \leq \delta.$$

Proof. We prove by induction on i that

$$\Pr_{h,\mathrm{Z}} \left[\exists j \leq i,\ M_j \neq \widetilde{M_j} \right] \leq \left(\delta_{\mathsf{N}} + w^2 2^{-\ell} \right) \cdot i.$$

The base case $i = 0$ is trivial. Fix some $i \geq 1$, and denote by E the "bad" set

$$E = \left\{ (h, \mathrm{Z}) : \exists j < i \text{ such that } M_j \neq \widetilde{M_j} \text{ or } h \notin \mathrm{GOOD}_{i-1,\mathrm{Z}} \right\}.[5]$$

Next, we write

$$\begin{aligned}
\Pr_{h,\mathrm{Z}} \left[\exists j \leq i, M_j \neq \widetilde{M_j} \right] &= \Pr[E] \Pr \left[\exists j \leq i, M_j \neq \widetilde{M_j} \mid E \right] + \Pr[\neg E] \Pr \left[M_i \neq \widetilde{M_i} \mid \neg E \right] \\
&\leq \Pr[E] + \Pr \left[M_i \neq \widetilde{M_i} \mid \neg E \right] \\
&\leq \Pr \left[\exists j < i, M_j \neq \widetilde{M_j} \right] + \Pr \left[h \notin \mathrm{GOOD}_{i-1,\mathrm{Z}} \right] + \Pr \left[M_i \neq \widetilde{M_i} \mid \neg E \right].
\end{aligned}$$

By the induction's hypothesis the first term is at most $\left(\delta_{\mathsf{N}} + w^2 2^{-\ell} \right) (i - 1)$, and by Equation (4) the second term is at most δ_{N}, so it suffices to show that

$$\Pr_{h,\mathrm{Z}} \left[M_i \neq \widetilde{M_i} \mid \forall j < i \quad M_j = \widetilde{M_j} \quad \text{and} \quad h \in \mathrm{GOOD}_{i-1,\mathrm{Z}} \right] \leq w^2 2^{-\ell}.$$

In fact, we shall show that for any *fixed* $\mathrm{Z}_1, \dots, \mathrm{Z}_{i-1}$ and h satisfying the above conditioning, we have

$$\Pr_{\mathrm{Z}_i} \left[M_i \neq \widetilde{M_i} \right] \leq w^2 2^{-\ell}.$$

Since $h \in \mathrm{GOOD}_{i-1,\mathrm{Z}}$,

$$\left\| M_{i-1}^{2^{r_1}} - \mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}} \left(M_{i-1}, h, 2^{r_1} \right) \right\|_\infty \leq \varepsilon_{\mathsf{N}}.$$

[5]For brevity, we use the notation $\mathrm{GOOD}_{i,\mathrm{Z}}$ even when the Z vector contains more than i elements, in which case we just ignore the rest of them.

Recall that we assumed that $\widetilde{M}_{i-1} = M_{i-1}$, and so

$$\left\| M_{i-1}^{2^{r_1}} - \widetilde{M}_{i-1}^{(2^{r_1})} \right\|_\infty \leq \varepsilon_{\mathsf{N}} = 2^{-2t}$$

as well. By Corollary 4.3, with probability at least $1 - w^2 2^{-\ell}$ over Z_i,

$$\left\lfloor M_{i-1}^{2^{r_1}} - \mathrm{Z}_i \cdot 2^{-2t} J_w \right\rfloor_t = \left\lfloor \widetilde{M}_{i-1}^{(2^{r_1})} - \mathrm{Z}_i \cdot 2^{-2t} J_w \right\rfloor_t,$$

which simply amounts to $M_i = \widetilde{M}_i$, as desired (see Equation (2) for the definition of M_i). This completes the inductive step. $\square$

Next, we handle the accuracy guarantee.

Claim 4.6. *For all $i \in \{0, 1, \dots, r_2\}$ and all Z it holds that*

$$\left\| M_i - A^{2^{ir_1}} \right\|_\infty \leq 2^{-t+1} w \sum_{j=0}^{i-1} 2^{jr_1}.$$

In particular, $\|M_{r_2} - A^n\|_\infty \leq \varepsilon$.

Proof. We prove the claim by induction on i. The base case follows since $M_0 = A$. Fix some $i \geq 1$. By the definition of M_i we have

$$\left\| M_i - M_{i-1}^{2^{r_1}} \right\|_{\max} \leq 2^{-t} + 2^{-2t+\ell} \leq 2^{-t+1},$$

and so by Claim 3.1, $\left\| M_i - M_{i-1}^{2^{r_1}} \right\|_\infty \leq 2^{-t+1} w$. Write

$$\left\| M_i - A^{2^{ir_1}} \right\|_\infty \leq \left\| M_i - M_{i-1}^{2^{r_1}} \right\|_\infty + \left\| M_{i-1}^{2^{r_1}} - \left(A^{2^{(i-1)r_1}} \right)^{2^{r_1}} \right\|_\infty.$$

By the induction's hypothesis and Claim 3.2, the second term is at most

$$2^{r_1} \cdot 2^{-t+1} w \sum_{j=0}^{i-2} 2^{jr_1}.$$

Overall, we get

$$\left\| M_i - A^{2^{ir_1}} \right\|_\infty \leq 2^{-t+1} w + 2^{r_1} \cdot 2^{-t+1} w \sum_{j=0}^{i-2} 2^{jr_1} \leq 2^{-t+1} w \sum_{j=0}^{i-1} 2^{jr_1}.$$

This completes the induction. The "In particular" part follows from our choice of parameters, noting that $2^{-t+1}w \cdot 2^r \leq \varepsilon$. □

Claim 4.5 tells us that with probability at least $1-\delta$, $\widetilde{M}_{r_2} = M_{r_2}$. By Claim 4.6 above, $\left\|\widetilde{M}_{r_2} - A^n\right\| \leq \varepsilon$, so the correctness follows.

For the space complexity, by Theorem 3.14, N takes $O\left(t + r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)$ space and the rest of the operations per iteration are absorbed within the latter term. This yields space complexity of $r_2 \cdot O\left(r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right) = r_2 \cdot O\left(\log \frac{nw}{\varepsilon\delta}\right)$. □

Given the above theorem, one can readily obtain a deterministic algorithm for matrix powering by averaging over all seeds, using space

$$O\left(r_2\ell + d_{\mathsf{N}} + \log \frac{nw}{\varepsilon\delta}\right) = O\left(r_2 \log \frac{nw}{\varepsilon\delta} + r_1^2 + r_1 \log \frac{nw}{\varepsilon\delta}\right).$$

Setting $r_1 = r_2 = \sqrt{r} = \sqrt{\log n}$, and $\delta = \varepsilon$, one gets $O(\varepsilon)$ approximation in the induced ℓ_∞ norm using space

$$O\left(\sqrt{\log n} \cdot \log \frac{nw}{\varepsilon}\right).$$

We omit the details as we take a different approach for this final step in our improved algorithm.

5 Approximate Powering in Small Space

In this section, we present our improvement upon the Saks–Zhou algorithm to obtain better space complexity for approximating large powers of matrices, following the outline given in Section 2.

5.1 The algorithm

Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ the desired confidence. Let $r \in \mathbb{N}$, and write $r = r_1 r_2$ for some $r_1, r_2 \in \mathbb{N}$ to be chosen later on. We set the accuracy and confidence of Nisan algorithm to be $\varepsilon_{\mathsf{N}} = 2^{-2r_1}$ and $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, respectively. Nisan's algorithm N will work with each entry represented with $t_1 = 4r_1 + \log w$ digits of precision. Following Theorem 3.14, the seed for Nisan's algorithm is of length

$$d_{\mathsf{N}} = O\left(r_1 \cdot \left(t_1 + r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)\right) = O\left(r_1^2 + r_1 \log \frac{r_2 w}{\delta}\right).$$

For the shift and truncate we take $\ell = \log \frac{2w^2 r_2}{\delta}$. Note that ℓ only depends on w and r_2, and not on n or ε. The number of bits required for the shifts is thus

$$r_2 \cdot \ell = O\left(r_2 \cdot \log \frac{r_2 w}{\delta}\right).$$

Finally, set $t_2 = \log \frac{16w^2 r_2 n}{\varepsilon\delta}$, and notice that $t_2 = \Omega(\log \frac{n}{\varepsilon})$. We stress that the key fact that unlike [SZ99], here we take $t_1 \ll t_2$.

The algorithm $\mathsf{SZ}_{\mathsf{Imp}}$ gets as input a stochastic matrix $A \in \mathbb{R}^{w\times w}$, $h \in \{0,1\}^{d_{\mathsf{N}}}$, and $(\mathrm{Z}_1, \ldots, \mathrm{Z}_{r_2}) \in \left\{0, \ldots, 2^{\ell} - 1\right\}^{r_2}$. Without loss of generality we may assume that each entry of the input matrix A is given with t_2 digits of precision. The algorithm proceeds as follows.

1. Set $\widetilde{M}_0 = A$.
2. For $i = 1, \ldots, r_2$,
   (a) Compute $\left(\widetilde{M}_{i-1}^{(1)}, \widetilde{M}_{i-1}^{(2)}, \ldots, \widetilde{M}_{i-1}^{(2^{r_1})}\right) = \mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}\left(\widetilde{M}_{i-1}, h, 2^{r_1}, t_1\right)$.
   (b) Compute $\widetilde{M}_i = \left\lfloor \mathsf{R}\left(\widetilde{M}_{i-1}^{(1)}, \ldots, \widetilde{M}_{i-1}^{(2^{r_1})}, \widetilde{M}_{i-1}, 3t_2\right) - \mathrm{Z}_i 2^{-2t_2} J_w \right\rfloor_{t_2}$.[6]
3. Output $\widetilde{M}_{r_2}$.

We first determine our algorithm's space complexity.

Lemma 5.1. *Computing* $\mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \mathrm{Z})$ *takes*

$$O\left((\log n + r_2 \log w) \cdot \log\log \frac{nw}{\varepsilon\delta} + r_2 \log \frac{1}{\delta} + r_2 \left(\log\log \frac{nw}{\varepsilon\delta}\right)^2\right)$$

space.

Proof. Consider the function $f(\widetilde{M}_i) = \widetilde{M}_{i+1}$ describing one iteration of Item 2. Note that this function has the same input and output length – a $w \times w$ matrix, and that each entry is represented by t_2 bits. The function f is the composition of three functions:

[6]The Richardson iteration may output a matrix which is not substochastic. This can be addressed by first rounding all negative entries to 0 and all entries larger than 1 to 1. This step can only improve the accuracy. Then, if the sum of entries in some row exceeds 1, decrease the largest entry in that row by the smallest value that will result in its sum being at most 1 (note that we may not be able to get the sum to be exactly 1 as we work with $O(t_2)$ bits of precision). In terms of accuracy, the above correction is negligible compared to the truncation step for a good (h, Z).

- The Nisan generator N: By Theorem 3.14, this takes

$$O\left(t_1 + \log t_2 + r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right) = O\left(\log\log \frac{n}{\varepsilon} + r_1 + \log \frac{w}{\delta}\right)$$

  space.

- Richardson Iteration: By Lemma 3.9, this takes

$$O\left(\log^2 t_2 + \log t_2 \cdot \log(2^{r_1} w t_2)\right) = O\left(\left(\log\log \frac{nw}{\varepsilon\delta}\right)^2 + \log(2^{r_1}w)\cdot \log\log \frac{nw}{\varepsilon\delta}\right)$$

  space.

- Truncation: Takes $O(\log t_2) = O\left(\log\log \frac{nw}{\varepsilon}\right)$ space.

The algorithm is a composition of f on itself r_2 times so by Corollary 3.4 we can sum the above and multiply by r_2, obtaining our desired overall space complexity. □

5.2 Proof of correctness

One can verify that by our choice of parameters above, that $\ell < t_2$, $t_1 < t_2$, and

$$\varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1} \leq \frac{1}{4(2^{r_1}+1)}, \tag{5}$$

$$2^{-t_2} \leq \frac{1}{2^{r_1}+1}. \tag{6}$$

We abbreviate $\mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}(A, h, 2^{r_1}, t_1)$ with $\mathsf{N}(A, h)$.

Theorem 5.2. *For any $w \times w$ stochastic matrix A and integers r_1, r_2, the above algorithm satisfies the following. With probability at least $1-\delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$ and $\mathrm{Z} = (\mathrm{Z}_1, \dots, \mathrm{Z}_{r_2}) \in \{0, \dots, 2^\ell - 1\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \mathrm{Z})$ satisfies*

$$\left\| A^n - \widetilde{M}_{r_2} \right\|_\infty \leq \varepsilon,$$

where $r = r_1 r_2$ and $n = 2^r$. Moreover, $\mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \mathrm{Z})$ runs in space

$$O\left((\log n + r_2 \log w)\cdot \log\log \frac{nw}{\varepsilon\delta} + r_2 \log \frac{1}{\delta} + r_2\left(\log\log \frac{nw}{\varepsilon\delta}\right)^2\right).$$

Proof. We let M_i be the "true" random truncation, similar to the analysis in Section 4. That is, $M_0 = A$, and for each $i \in [r_2]$,

$$M_i(\mathrm{Z}) = \left\lfloor M_{i-1}(\mathrm{Z})^{2^{r_1}} - \mathrm{Z}_i \cdot 2^{-2t_2} J_w \right\rfloor_{t_2}. \tag{7}$$

Observe, again, that the M_i-s do not depend on h. For brevity, we omit the dependence on h and Z whenever it is clear from context.

Next, we argue that with high probability (over h and the Z-s), $\widetilde{M}_i = M_i$. Toward this end, we similarly define for each fixing of Z,

$$\mathrm{GOOD}_{i,\mathrm{Z}} = \left\{ h \in \{0,1\}^{d_{\mathsf{N}}} : \forall j \leq 2^{r_1}, \left\| M_i^j - M_i^{(j)} \right\|_\infty \leq \varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1} \right\}, \tag{8}$$

where $\left(M_i^{(1)}, \dots, M_i^{(2^{r_1})}\right) = \mathsf{N}(M_i, h)$. (Note that here we feed N with M_i and not $\widetilde{M}_i$, similar to what we did in Section 4.) By Theorem 3.14, we get that for any $i \in [r_2]$ and $\mathrm{Z} = (\mathrm{Z}_1, \dots, \mathrm{Z}_i)$,

$$\Pr_{h \in \{0,1\}^{d_{\mathsf{N}}}} [h \in \mathrm{GOOD}_{i,\mathrm{Z}}] \geq 1 - \delta_{\mathsf{N}}. \tag{9}$$

Claim 5.3. *It holds that*

$$\Pr_{h,\mathrm{Z}} \left[\exists k \in [r_2],\ M_k \neq \widetilde{M}_k \right] \leq \left(\delta_{\mathsf{N}} + w^2 2^{-\ell} \right) r_2 \leq \delta.$$

Proof. The proof is similar to the proof of Claim 4.5. We prove by induction on i that

$$\Pr_{h,\mathrm{Z}} \left[\exists k \leq i, M_k \neq \widetilde{M}_k \right] \leq \left(\delta_{\mathsf{N}} + w^2 2^{-\ell} \right) \cdot i.$$

The base case $i = 0$ is trivial. Fixing some $i \geq 1$, we denote by E the set

$$E = \left\{ (h, \mathrm{Z}) : \exists k < i \text{ such that } M_k \neq \widetilde{M}_k \text{ or } h \notin \mathrm{GOOD}_{i-1,\mathrm{Z}} \right\},$$

and again we have

$$\begin{aligned}
\Pr_{h,\mathrm{Z}} \left[\exists k \leq i, M_k \neq \widetilde{M}_k \right] &= \Pr[E] \Pr \left[\exists k \leq i, M_k \neq \widetilde{M}_k \mid E \right] + \Pr[\neg E] \Pr \left[M_i \neq \widetilde{M}_i \mid \neg E \right] \\
&\leq \Pr[E] + \Pr \left[M_i \neq \widetilde{M}_i \mid \neg E \right] \\
&\leq \Pr \left[\exists k < i, M_k \neq \widetilde{M}_k \right] + \Pr \left[h \notin \mathrm{GOOD}_{i-1,\mathrm{Z}} \right] + \Pr \left[M_i \neq \widetilde{M}_i \mid \neg E \right].
\end{aligned}$$

By the induction's hypothesis the first term is at most $\left(\delta_{\mathsf{N}} + w^2 2^{-\ell} \right) \cdot (i-1)$, and by Equation (9) the second term is at most δ_{N}. Thus, it suffices to show that

$$\Pr_{h,\mathrm{Z}}\left[M_i \neq \widetilde{M_i} \;\middle|\; \forall k < i,\, M_k = \widetilde{M_k} \text{ and } h \in \mathrm{GOOD}_{i-1,\mathrm{Z}}\right] \leq w^2 2^{-\ell}.$$

We show that for any fixed $\mathrm{Z}_1, \dots, \mathrm{Z}_{i-1}$ and h satisfying the conditioning, we have

$$\Pr_{\mathrm{Z}_i}\left[M_i \neq \widetilde{M_i}\right] \leq w^2 2^{-\ell}.$$

Since $h \in \mathrm{GOOD}_{i-1,\mathrm{Z}}$, for all $j \leq 2^{r_1}$ we have that

$$\left\|M_{i-1}^{j} - M_{i-1}^{(j)}\right\|_\infty \leq \varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1}.$$

Recall that we assume that $\widetilde{M}_{i-1} = M_{i-1}$, and so for all $j \leq 2^{r_1}$,

$$\left\|M_{i-1}^{j} - \widetilde{M}_{i-1}^{(j)}\right\|_\infty \leq \varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1}.$$

Using Lemma 3.9 and the guarantee of Equation (5), we get

$$\left\|\mathsf{R}\left(\widetilde{M}_{i-1}^{(1)}, \dots, \widetilde{M}_{i-1}^{(2^{r_1})}, \widetilde{M}_{i-1}, 3t_2\right) - M_{i-1}^{2^{r_1}}\right\|_\infty \leq (2^{r_1} + 1) \cdot 2^{-3t_2} \leq 2^{-2t_2}.$$

Thus, by Corollary 4.3, with probability at least $1 - w^2 2^{-\ell}$ over Z_i,

$$\left\lfloor M_{i-1}^{2^{r_1}} - \mathrm{Z}_i \cdot 2^{-2t_2} J_w \right\rfloor_{t_2} = \left\lfloor \mathsf{R}\left(\widetilde{M}_{i-1}^{(1)}, \dots, \widetilde{M}_{i-1}^{(2^{r_1})}, \widetilde{M}_{i-1}, 3t_2\right) - \mathrm{Z}_i \cdot 2^{-2t_2} J_w \right\rfloor_{t_2},$$

which simply means that $M_i = \widetilde{M_i}$, recalling the definition of M_i from Equation (7). This completes the inductive step. $\square$

For the accuracy guarantee, we have

Claim 5.4. *For all $i \in \{0, 1, \dots, r_2\}$ and all Z it holds that*

$$\left\|M_i - A^{2^{ir_1}}\right\|_\infty \leq 2^{-t_2+1} w \sum_{j=0}^{i-1} 2^{jr_1}.$$

In particular, $\|M_{r_2} - A^n\|_\infty \leq \varepsilon$.

The proof is identical to the proof of Claim 4.6, so we omit it. To conclude, note that by Claim 5.3 we have that with probability at least $1-\delta$, $\widetilde{M}_{r_2} = M_{r_2}$, and by Claim 5.4 we establish the accuracy guarantee $\left\|\widetilde{M}_{r_2} - A^n\right\| \leq \varepsilon$. The space requirement was established in Lemma 5.1. □

5.3 A high-accuracy deterministic approximation

The dependence of Theorem 5.2 on ε is only double-logarithmic, and so taking a tiny ε does not deteriorate the space complexity by much. The dependence on δ, however, is logarithmic. When we fix

$$r_1 = r_2 = \sqrt{\log n}$$
$$\varepsilon, \delta \geq \frac{1}{n},$$

in Theorem 5.2, we get space complexity $\tilde{O}(\log n + \sqrt{\log n} \cdot \log \frac{w}{\delta})$. This means that to get space complexity $\tilde{O}(\log n + \sqrt{\log n} \cdot \log w)$ we cannot take δ much smaller than $\frac{1}{w}$.

Now suppose our goal is to get a *deterministic* algorithm approximating A^n to within $\frac{1}{n}$ accuracy. We can follow [SZ99] and by averaging over all offline seeds (namely, h and the Z-s), taking $\delta = \frac{1}{w}$, get a deterministic approximation with $\frac{1}{w}$ error. However, in this section we show how to get a much better accuracy $\frac{1}{n}$. Our algorithm is simple. Instead of *averaging* over all the good and bad offline randomness strings, we iterate the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm over all (h, Z)-s and take the entry-wise *median* of the outputs. This approach only requires $\delta = \Omega(1)$ and works because we know more than half of the offline strings are good.

Formally, given a set of matrices $\{A_1, \ldots, A_m\}$, we denote by $\mathrm{median}_{i\in[m]} A_i$ the matrix M for which $M[a,b]$ is the median of $A_i[a,b]$ over all $i \in [m]$. The algorithm $\mathsf{SZ}^+_{\mathsf{Imp}}$ proceeds as follows. We are given a stochastic matrix $A \in \mathbb{R}^{w\times w}$, $n \in \mathbb{N}$, and a desired accuracy parameter $\varepsilon > 0$. Set $r_1 = r_2 = \sqrt{\log n}$, $\delta_{\mathsf{SZ}_{\mathsf{Imp}}} = \frac{1}{4}$, and $\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} = \frac{1}{8w(n+1)\log n}$. For this choice of parameters, the truncation parameter t_2 from Section 5 satisfies

$$t_2 = \log \frac{16w^2 r_2 n}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}} = O\left(\log nw\right).$$

Also, note that d_{N} in $\mathsf{SZ}_{\mathsf{Imp}}$ satisfies

$$d_{\mathsf{N}} = O\left(r_1^2 + r_1 \cdot \log \frac{r_2 w}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right) = O\left(\log n + \sqrt{\log n} \cdot \log w\right),$$

and the number of bits needed to represent $\mathrm{Z}_1, \ldots, \mathrm{Z}_{r_2}$ is given by

$$|\mathrm{Z}| = O\left(r_2 \cdot \log \frac{r_2 w}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right) = O\left(\sqrt{\log n} \cdot \log\log n + \sqrt{\log n} \cdot \log w\right).$$

Then,

1. For $i = 1, \ldots, \log n$, compute
$$\widetilde{M}_{2^i} = \underset{h,\mathrm{Z}}{\text{median}} \ \mathsf{SZ}_{\mathsf{Imp}}\left(\lfloor A \rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \mathrm{Z}\right),$$
where we take the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm with accuracy $\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}$ and confidence $\delta_{\mathsf{SZ}_{\mathsf{Imp}}}$.[7]

2. For $j \in [n]$, we let $b_{i,j} \in \{0,1\}$ be such that $j = \sum_{i=0}^{\lceil \log n \rceil} b_{i,j} 2^i$ is the binary representation of j. Compute
$$\widetilde{M}_j = \prod_{i : b_{i,j}=1} \widetilde{M}_{2^i}.$$

3. Output $\widetilde{M} = \mathsf{R}\left(\widetilde{M}_1, \ldots, \widetilde{M}_n, A, k\right)$ for $k = \lceil \log \frac{n}{\varepsilon} + 1 \rceil$.

Theorem 5.5. *Given a $w \times w$ stochastic matrix A, the algorithm $\mathsf{SZ}^+_{\mathsf{Imp}}$ above satisfies*
$$\left\| A^n - \widetilde{M} \right\|_\infty \le \varepsilon,$$
and runs in space
$$O\left(\left(\log n + \sqrt{\log n} \cdot \log w\right) \cdot \log\log nw + \left(\log\log \frac{1}{\varepsilon}\right)^2 + \log\log \frac{1}{\varepsilon} \cdot \log nw\right).$$
In particular, for $\varepsilon = 2^{-\operatorname{polylog}(n)}$, the space complexity is $\widetilde{O}\left(\log n + \sqrt{\log n} \cdot \log w\right)$.

Proof. First, note that for each i, $\left|\widetilde{M}_{2^i}\right| = O(w^2 t_2)$, and so
$$\left|\widetilde{M}_j\right| = O\left(w^2 \left(t_2 + \log w\right) \log n\right) = O(w^2 t_2 \log n).$$

We start by analyzing the space complexity.

- Following Theorem 5.2, the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm with the prescribed parameters takes
$$O\left((\log n + r_2 \log w) \log\log \frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2 \log \frac{1}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2 \left(\log\log \frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right)^2\right)$$

[7]To be completely accurate, the second and third arguments to $\mathsf{SZ}_{\mathsf{Imp}}$ must be integers and so are taken to be $\lfloor \sqrt{i} \rfloor$ rather than $\sqrt{i}$. This then yields an approximation to the $2^{\lfloor \sqrt{i} \rfloor^2}$ power of A. As $i - \lfloor \sqrt{i} \rfloor^2 \le 2\sqrt{i} + 1$, computing the "missing" $2^{O(\sqrt{i})}$ power can be done in an iterative manner (for $\approx \log\log i = \log\log\log n$ iterations), and without effecting the overall space complexity.

space, which is

$$O\left(\left(\log n + \sqrt{\log n} \cdot \log w\right) \cdot \log\log nw\right),$$

and running it for $\log n$ times requires only an additional counter of $\log\log n$ bits.

- Computing the median of m numbers $a_1, \dots, a_m$ each represented via t bits can be done in $O(\log m + \log t)$ space. E.g., for a fixed number a_j, we can go over all a_i-s and count how many of them are smaller than a_j breaking ties lexicographically, i.e.,

  $$a_i \prec a_{i'} \iff (a_i < a_{i'}) \vee ((a_i = a_{i'}) \wedge (i < i')).$$

  In our case, this amount to space

  $$d_{\mathsf{N}} + |\mathrm{Z}| + O\left(\log t_2 w^2\right) = O\left(\log n + \sqrt{\log n} \cdot \log w\right).$$

- Computing the powers in Item 2 takes

  $$O\left(\log\log n \cdot \log(t_2 w^2)\right) = O\left(\log\log n \cdot \log(w \log n)\right)$$

  space.

- Applying R takes

  $$O\left(\left(\log\log\frac{n}{\varepsilon}\right)^2 + \log\log\frac{n}{\varepsilon} \cdot \log\left((n+1) \cdot w^2 t_2 \log n\right)\right)$$

  space, following Lemma 3.9, which is

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

Our algorithm is essentially a composition of the above procedures, and so the claim on the space complexity follows from composition of space-bounded algorithms (Claim 3.3).

We now proceed with the correctness. By Theorem 5.2, for at least $\frac{3}{4}$ of the (h, Z)-s, we have

$$\left\|\mathsf{SZ}_{\mathsf{Imp}}(\lfloor A\rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \mathrm{Z}) - \lfloor A\rfloor_{t_2}^{2^i}\right\|_{\max} \le \left\|\mathsf{SZ}_{\mathsf{Imp}}(\lfloor A\rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \mathrm{Z}) - \lfloor A\rfloor_{t_2}^{2^i}\right\|_{\infty} \le \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}},$$

and so for at least $\frac{3}{4}$ of the (h, Z)-s we get that for all $(a, b) \in [w]^2$,

$$\left| \mathsf{SZ}_{\mathsf{Imp}} \left(\lfloor A \rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \mathrm{Z} \right) [a, b] - \lfloor A \rfloor_{t_2}^{2^i} [a, b] \right| \leq \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$

Thus, for all $(a, b) \in [w]^2$,

$$\left| \left(\underset{h,\mathrm{Z}}{\mathrm{median}}\ \mathsf{SZ}_{\mathsf{Imp}} \left(\lfloor A \rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \mathrm{Z} \right) \right) [a, b] - \lfloor A \rfloor_{t_2}^{2^i} [a, b] \right| \leq \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$

This is true for all indices (a, b) and all $i \in [\log n]$ and so by Claim 3.1, for all $i \in [\log n]$,

$$\left\| \widetilde{M}_{2^i} - \lfloor A \rfloor_{t_2}^{2^i} \right\|_\infty \leq w \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$

By Claim 3.2, $\left\| A^j - \lfloor A \rfloor_{t_2}^j \right\|_\infty \leq j w 2^{-t_2}$, and applying Claim 3.2 again for multiplication of $\log n$ matrices, we get that for every $j \leq n$,

$$\left\| \widetilde{M}_j - A^j \right\|_\infty \leq \left\| \widetilde{M}_j - \lfloor A \rfloor_{t_2}^j \right\|_\infty + \left\| \lfloor A \rfloor_{t_2}^j - A^j \right\|_\infty \leq \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} w \log n + n w 2^{-t_2} \leq \frac{1}{4(n+1)}.$$

Using Lemma 3.9, we obtain

$$\left\| \mathsf{R}(\widetilde{M}_1, \ldots, \widetilde{M}_n, A, k) - A^n \right\|_\infty \leq (n+1) \cdot 2^{-k} \leq \varepsilon,$$

which completes the proof. □

6 Approximating the Iterated Product

In this section, we prove the following theorem which implies Theorem 1.1.

Theorem 6.1. *For any $n, w \in \mathbb{N}$ where $n \geq w$, and any $\varepsilon > 0$, there exists a deterministic algorithm that given $w \times w$ stochastic matrices $A_1, \ldots, A_n$, approximates $A_1 \cdots A_n$ to within error ε in space*

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

6.1 Laying the groundwork

Now that our matrix powering algorithm has been established, we develop some of the ideas, discussed informally in Section 2.2, in preparation for our complete IMM algorithm.

6.1.1 Improving the dependence on the confidence parameter

Recall that the seed length and space complexity of the randomized IMM algorithm induced by the Nisan generator have poor dependence on the confidence parameter δ. The discussion in Section 2.2 shows that the confidence parameter has to be smaller than $\frac{1}{n}$. This requires us to use a PRG with a better dependence on the confidence parameter.

Theorem 6.2. *There exists an algorithm $\Lambda_{\varepsilon,\delta}$ that gets as input:*

1. *A sequence of $w \times w$ substochastic matrices $(A) = (A_1, \ldots, A_n)$ in which every entry is represented using at most s bits.*
2. *An accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and a canonicalization parameter $t \in \mathbb{N}$, where $t \leq s$.*
3. *A seed $h \in \{0,1\}^{d_\Lambda}$ of length $d_\Lambda = \left(\log n \cdot \left(t + \log \frac{nw}{\varepsilon}\right) + \log\log n \cdot \log \frac{1}{\delta}\right)$.*

The algorithm runs in space $O\left(\log s + t + \log \frac{nw}{\varepsilon} + \log\log \frac{1}{\delta}\right)$ and outputs the matrix sequence

$$(M_h) = \Lambda_{\varepsilon,\delta}((A), h, n, t),$$

and satisfies the following. With probability at least $1 - \delta$ over $h \in \{0,1\}^{d_\Lambda}$, it holds that for all $1 \leq i < j \leq n$,

$$\left\|(M_h)_{i,j} - A_i \cdots A_j\right\|_\infty \leq \varepsilon + nw \cdot 2^{-t}.$$

When we omit the parameter t, we implicitly set $t = s$, and then the error guarantee is simply ε.

Comparing Theorem 6.2 with Theorem 3.14, we see that Theorem 6.2 improves the dependence on δ both in the the space complexity and in the seed length, d_Λ. The construction of $\Lambda_{\varepsilon,\delta}$ starts with Nisan's PRG with *constant* confidence, and amplifies its confidence to the desired δ using a sampler, via "Armoni's sampler trick" [Arm98]. We prove Theorem 6.2 in Appendix C, where we also discuss the underlying technique.

6.1.2 Dealing with the shifts

As discussed in Section 2.2, the shifts require new ideas and substantial effort. Our first attempt is the following algorithm, wherein $t_1 = \Theta(\log w)$, $t_2 = \Theta(\log n)$, $r_1 r_2 = \log n$ (and for simplicity, say, $r_1 = r_2 = \sqrt{\log n}$).

1. Shift the entry of each of the input matrices by $Z \cdot 2^{-2t_2}$ where $Z \sim \{0, 1, \ldots, 2^{t_2} - 1\}$.

2. For $i = 1, \dots, r_2$,
   (a) Partition the iterated product to sub-products, each consists of 2^{r_1} matrices.
   (b) Truncate the matrices to precision t_1 and use $\Lambda_{\varepsilon_\Lambda, \delta_\Lambda}$ to approximate the iterated sub-products.
   (c) Regain the high accuracy via the Richardson iteration, and then truncate to precision t_2.

Note that as in the powering algorithm, at each level we truncate the input to t_1 bits of accuracy, where $t_1 = \Theta(\log w + \sqrt{\log n})$, apply $\Lambda_{\varepsilon_\Lambda, \delta_\Lambda}$, and then use Richardson iteration to recover t_2 bits of accuracy, where recall $t_2 = \Theta(\log n)$. The role of the "outer" rounding, in (b), is to decorrelate the randomness h from the output, and at this stage it is not clear whether this step achieves this goal. Notice also that unlike in the powering algorithm, we shift *the input*, and we shift all A_i-s by the same shift, using $O(\log n)$ bits for that single shift. There are no other shifts for intermediate levels in the algorithm. Our hope is that investing $O(\log n)$ bits of randomness in this initial single shift "takes care" of all future iterations.

The analysis of this first attempt boils down to algebraically expressing how a shift of the input affects the output product, which we accomplish in Section 6.3. In Lemma 6.6 we prove that a shift ζ of each entry of $A_1, \dots, A_n$ results in an error matrix $E(\zeta)$, where $0 \le E(\zeta) \le \zeta \cdot T$, inequalities are entry-wise, and the matrix T is defined by

$$T = J_w \sum_{k=1}^{n} A_{k+1} \cdots A_n. \tag{10}$$

This implies that each entry of E has magnitude at most $n^2\zeta$. However, generalizing the truncation lemma, Lemma 4.2, to the case where the shifts are given by some error function $E(\zeta)$ reveals that we need to bound $E(\zeta)$ not only from above, but also from below. We give the precise details in Lemma 6.7. Luckily for us, it turns out that

$$0 < (1 - wn\zeta)\zeta \cdot T \le E(\zeta) \le \zeta \cdot T,$$

and that with high probability, a ζ-shift of the input is good, in the sense that the output is far from the boundary of a truncation. In particular, we conclude that at least in the first iteration, with high probability over the shift, the truncation indeed decorrelates h from the output. We give the precise details in Section 6.3.

Furthermore, by taking the union bound over all the true[8] matrices that are obtained

[8]The "true" matrices will be defined similarly to the $M_i(\mathrm{Z})$ in Section 5.2, wherein we apply the random truncation and the Richardson iteration on the true products.

as partial products in the computation, we see that with high probability (over the initial shift) all these products are *ρ-safe*, in the sense that their entries are at least ρ-far from a 2^{-t_2} boundary, for ρ and 2^{-t_2} that may be polynomially-small in n. Thus, if we could approximate the correct matrices with accuracy better than, say, $\rho/2$, then that approximation is also $\rho/2$ safe, and a truncation to t_2 bits of accuracy gives a pre-determined result, independent of h.

However, the main challenge in the analysis is that we need to track the shift effects not only upon multiplication, but also upon the truncation steps that we have throughout the computation. Here the approach runs into an unexpected problem: How should we choose the parameter ρ? Clearly, ρ should be smaller than 2^{-t_2} (as we want to be ρ-far from a 2^{-t_2} boundary). But when we truncate to t_2 bits of accuracy, we introduce an error of 2^{-t_2}, and so $\rho \geq 2^{-t_2}$. Indeed, after the truncation to t_2 bits of accuracy, we are *always* at a 2^{-t_2} boundary point, and therefore the approximated matrix that we get is never safe no matter what shift we choose.

To summarize, there are two contradicting forces in our strategy: (1) perturbing the input, and (2) the truncation. While the initial perturbation makes all *correct* iterated products safe, the truncation makes the *approximated* matrices unsafe. Perhaps a natural approach is to allow a deterioration in the truncation parameters, namely make t_2 smaller as the algorithm progresses. However, this does not work either because the argument seemingly loses $\log \frac{1}{\rho}$ bits of precision at each iteration, which is roughly $\log n$.

Our solution to the problem is to introduce *another* Richardson iteration step in order to make ρ smaller than 2^{-t_2}. The fact that we use *two* Richardson steps at each layer may look perplexing at first, but the utility of the two Richardson steps can be simply explained: The inner Richardson iteration, combined with the truncation performed right after, is designed to decorrelate h, whereas the outer Richardson iteration maintains a small universal error ρ *independent of the inner decorrelation procedure*. Thus, while the matrix after the truncation is not safe, the outer Richardson iteration brings it closer to the correct value – so close that it must be safe.

6.2 The algorithm

The algorithm partitions the input matrices of the iterated product into groups of size 2^{r_1}, approximates the product of the matrices in each group, and recurses. This defines a tree of depth $r_2 = \frac{r}{r_1}$ and arity 2^{r_1} where the $n = 2^r$ inputs are the leaves. Every level of the tree is a matrix sequence: the bottom level consists of 2^r matrices, the level above it consists of 2^{r-r_1} matrices, and in general the i-th level consists of $2^{r-r_1 i}$ matrices.

The above scheme calls for the following useful notation of matrices sequences: (M)

denotes a sequence of matrices, namely

$$(M) = ((M)_1, (M)_2, \ldots, (M)_m),$$

for m being the length of (M). We stress that the notation (M_i) means a matrix sequence *named* M_i, and so $(M_i)_j$ is the j-th matrix in the sequence M_i. Given $B = (b_1, b_2, \ldots)$ define

$$(M)_B \triangleq ((M)_{b_1}, (M)_{b_2}, \ldots), \text{and,}$$
$$\prod_B (M) \triangleq \prod_j (M)_{b_j}.$$

Let (M_i) denote the matrix sequence in level i[9], and consider its j-th matrix $(M_i)_j$. For simplicity, let us assume for now that the computation is exact, i.e., every node equals exactly the iterated product of its children. We thus have the following simple recursive relation: The product of a node's descendants equals the product of the iterated products associated with its immediate children. Algebraically, denote by $L_{i,j}$ as the set of leaves (or, matrix indices) below the j-th node of the i-th level, and by $\Gamma_{i,j}$ the indices in (M_{i-1}) that are directly below the j-th node in the i-level. This notation allows us to establish the following relation:

$$\prod_{L_{i,j}} (M_0) = \prod_{k \in \Gamma_{i,j}} \prod_{L_{i-1,k}} (M_{i-1}). \tag{11}$$

One can work out the exact set $L_{i,j}$ and $\Gamma_{i,j}$. Concretely, for all $i \in \{0, \ldots, r_2\}$ and $j \in \left\{1, \ldots, 2^{r_1(r_2-i)}\right\}$,

$$L_{i,j} = \left\{(j-1)2^{ir_1} + 1, (j-1)2^{ir_1} + 2, \ldots, j2^{ir_1}\right\},$$
$$\Gamma_{i,j} = \left\{k + (j-1)2^{r_1} : 1 \le k \le 2^{r_1}\right\}.$$

Note that the indexing of $L_{i,j}$ is with respect to the bottom sequence (M_0), and $\Gamma_{i,j}$ is with respect to the sequence (M_{i-1}). Also, $\Gamma_{i,j}$ does not really depend on j (although i does define the range in which the values of j are valid), and we only consider $\Gamma_{i,j}$ with $i \ge 1$.

We are now ready to describe our IMM algorithm. The algorithm $\mathsf{SZ}_{\mathsf{IMM}}$ gets as input a sequence of stochastic matrices $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$. Without loss of generality, we may assume that each entry of the input matrix A is given with t_2 digits of precision, where $t_2 = 12 \log n$. Set $r_1 = r_2 = \sqrt{\log n}$, $\ell = \frac{t_2}{2} - 1$ and instantiate the powering algorithm $\Lambda_{\varepsilon_\Lambda, \delta_\Lambda}$ with the following parameters: accuracy $\varepsilon_\Lambda = 2^{-2r_1}$, confidence $\delta_\Lambda = \frac{1}{n^3}$, and canonicalization parameter $t_1 = 4r_1 + \log w$. The algorithm proceeds as follows.

1. For $j = 1, \ldots, n$, set $(\widetilde{M_0})_j = A_j - \mathrm{Z} \cdot 2^{-2t_2} J_w$, where $\mathrm{Z} \sim \left\{0, 1, \ldots, 2^\ell - 1\right\}$.

[9]Levels are counted bottom-up and so the leaves are at level 0.

2. For $i = 1, \ldots, r_2$,

   (a) For $j = 1, \ldots, \frac{n}{2^{ir_1}}$, compute

$$(\widetilde{M}_i)_j = \mathsf{R}\bigg(\Big\lfloor \mathsf{R}\Big(\Lambda_{\varepsilon_\Lambda, \delta_\Lambda}((\widetilde{M}_{i-1})_{\Gamma_{i,j}}, h, 2^{r_1}, t_1), (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 6t_2\Big)\Big\rfloor_{t_2}, (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 8t_2\bigg).$$

3. Output $\widetilde{M}_{r_2}$.

The parameters of $\mathsf{SZ}_{\mathsf{IMM}}$ are chosen so that they satisfy the following inequalities:

$$\varepsilon_\Lambda + w \cdot 2^{r_1 - t_1} \leq \frac{1}{4(2^{r_1}+1)}, \tag{12}$$

$$2^{-t_2} \leq \frac{1}{4w(2^{r_1}+1)}, \tag{13}$$

so that we can apply the Richardson iteration. Also, assume $w \leq n$ since this is our regime of interest. In Section 6.4, we prove:

Theorem 6.3. *For any sequence of substochastic matrices $(M) = (M_1, \ldots, M_{2^r}) \in \mathbb{R}^{w \times w}$,*

$$\Pr_{h,\mathrm{Z}}\left[\left\|\mathsf{SZ}_{\mathsf{IMM}}((M), h, \mathrm{Z}) - M_1 \cdots M_{2^r}\right\|_\infty \geq \frac{1}{n^{12}}\right] \leq \frac{2}{n^3}.$$

For the space complexity of the algorithm, we have:

Lemma 6.4. *For $n \geq w$, the algorithm $\mathsf{SZ}_{\mathsf{IMM}}$ can be implemented in*

$$O\Big(\Big(\log n + \sqrt{\log n} \cdot \log w\Big) \cdot \log\log n\Big)$$

space, and using $O(\sqrt{\log n} \cdot \log w)$ random bits.

We prove the above lemma in Section 6.5. The deterministic algorithm given in Theorem 6.1 is obtained from Theorem 6.3 and Lemma 6.4 by employing another Richardson iteration:

Proof of Theorem 6.1. First, to get a deterministic algorithm that computes $A_1 \cdots A_n$, we average over all possible h, Z. Following Lemma 6.4, this can be done in space

$$O\Big(\Big(\log n + \sqrt{\log n} \cdot \log w\Big) \cdot \log\log n\Big),$$

recalling that

$$|\mathrm{Z}| + |h| = O\left(\log n \cdot \log\log n + \sqrt{\log n} \cdot \log w\right).$$

By Theorem 6.3, the resulting accuracy is at most $\frac{1}{n^{12}} + \frac{2}{n^3} \leq \frac{1}{4(n+1)}$. Clearly, we can approximate any sub-product $A_i \cdots A_j$ with the same parameters.

To obtain accuracy $\varepsilon > 0$, we apply the Richardson iteration, Lemma 3.9, with $k = O(\log \frac{n}{\varepsilon})$. The procedure R takes

$$O\left(\log^2 k + \log k \cdot \log \frac{n}{\varepsilon}\right) = O\left(\left(\log\log \frac{1}{\varepsilon}\right)^2 + \log n \cdot \log\log \frac{n}{\varepsilon}\right)$$

space, noting that we can assume without loss of generality that T, the maximum bit-complexity of our input, is $\mathrm{poly}(w, n, \log(1/\varepsilon))$. The theorem finally follows from composition of space-bounded algorithm. □

6.3 Perturbing the input

Before proceeding to the analysis of our algorithm, which will be given in Section 6.4 and Section 6.5, we turn our focus to the effect of shifting the input. Suppose we want to compute the product $A_1 \cdots A_n$. We shift every matrix A_i by ζ, entry-wise, and compute the perturbed product

$$\prod_{i=1}^{n} (A_i - \zeta J_w).$$

To analyze the effect of the initial shifting, we consider a robust notion of rounding:

Definition 6.5. *We say $z \in \mathbb{R}$ is (t, ρ)-dangerous if z is ρ-close to a* positive *multiple of 2^{-t}, i.e., if there exists a* positive *integer $n > 0$ such that $|z - n2^{-t}| \leq \rho$. Otherwise, we say z is (t, ρ)-safe. Also, we say a matrix $M \in \mathbb{R}^{w \times w}$ is (t, ρ)-dangerous if one of its entries is (t, ρ)-dangerous, and otherwise we say it is (t, ρ)-safe.*

Notice that a number $z \in \mathbb{R}$ is (t, ρ)-safe if its ρ-neighborhood is always truncated to the same number when we truncate it to precision of t digits.

Remark 1. *Numbers that are very close to 0 are safe by definition. Also, negative values are always truncated to zero, and so there are no boundary issues around 0.*

The terminology of (t, ρ)-dangerous is used in the original work of Saks and Zhou [SZ99]. We remark, however, that this terminology is not required for the Saks–Zhou algorithm as presented in Section 4, while it is essential for our IMM algorithm.

In the powering algorithm of Section 5 we shift *output* matrices whereas for our IMM algorithm we shift the *input* matrices. This yields a more complex behaviour. Let $E_{i,j}$

denote the difference between the true product and the shifted product at the $[i, j]$-th cell,

$$E_{i,j}(\zeta) \triangleq (A_1 \cdots A_n)[i,j] - \left(\prod_{i=1}^{n}(A_i - \zeta J_w)\right)[i,j].$$

Also, let $E(\zeta)$ be the $w \times w$ matrix, whose (i, j)-th entry is $E_{i,j}(\zeta)$. The following is an explicit formula for E.

Lemma 6.6. *Let $A_1, \dots, A_n \in \mathbb{R}^{w\times w}$ be stochastic, and J_w the all-ones matrix. Then,*

$$\prod_{i=1}^{n}(A_i - \zeta J_w) = A_1 \cdots A_n - \zeta J_w \sum_{k=1}^{n}(1 - w\zeta)^k A_{k+1} \cdots A_n.$$

In particular, $E(\zeta) = \zeta J_w \sum_{k=1}^{n}(1 - w\zeta)^k A_{k+1} \cdots A_n$.

Proof. Expanding the shifted product, we get a summation in which each term is of the form

$$(-\zeta)^{j-1} \cdot A_1 \cdots A_{i_1} J_w A_{i_1+2} \cdots A_{i_2} J_w \cdots A_{i_j} J_w A_{i_j+2} \cdots A_n$$

for some $j \in \{1, \dots, n\}$ and $1 \le i_1 < \cdots < i_j \le n-1$ (if $i_j = n-1$, we take $A_{i_j+2} \cdots A_n = I$). For any stochastic matrix A we have that $AJ_w = J_w$, and also $J_w^j = w^{j-1} J_w$ so the above, for $j \ge 2$, simplifies to

$$(-\zeta)^{j-1} J_w^{j-1} A_{i_j+2} \cdots A_n = (-\zeta)^{j-1} w^{j-2} J_w A_{i_j+2} \cdots A_n.$$

We sum over the above terms according to k – the last index of an all-ones matrix term, and j – the number of J_w instances picked, excluding the last one. We then get that

$$\begin{aligned}\prod_{i=1}^{n}(A_i - \zeta J_w) &= A_1 \cdots A_n + \sum_{k=1}^{n}\sum_{j=0}^{k-1}\sum_{1\le i_1<i_2<\cdots<i_j<k}(-\zeta)^{j+1} w^j J_w A_{k+1} \cdots A_n \\ &= A_1 \cdots A_n - \zeta \sum_{k=1}^{n}\sum_{j=0}^{k-1}\binom{k-1}{j}(-w\zeta)^j J_w A_{k+1} \cdots A_n \\ &= A_1 \cdots A_n - \zeta J_w \sum_{k=1}^{n}(1 - w\zeta)^{k-1} A_{k+1} \cdots A_n.\end{aligned}$$

□

It follows from Lemma 6.6 that

$$T[i,j] \cdot \zeta \cdot (1 - wn\zeta) \le E_{i,j}(\zeta) \le T[i,j] \cdot \zeta,$$

where $T = J_w \sum_{k=1}^{n} A_{k+1} \cdots A_n$ is as in Equation (10). Furthermore,

$$0 \leq (A_1 \cdots A_n)[i,j] \leq T[i,j] \leq n^2.$$

Thus, if $T[i,j]$ is very small then so is the true product $(A_1 \cdots A_n)[i,j]$, and in this case we will truncate to zero.

The next lemma is a generalization of the truncation lemma, Lemma 4.2, to the case where the shifts are given by some function (e.g., $E_{i,j}$), with a fairly controlled behaviour, rather than by some fixed scalar.

Lemma 6.7. *Let $t, \ell, C \in \mathbb{N}$ satisfying $0 \leq C < 2^{t-\ell}$, and define $D = 2^{-2t}\{0, 1, \ldots, 2^{\ell} - 1\}$. Let $e \colon D \to \mathbb{R}$ be such that for all $\zeta \in D$,*

$$C(1 - 2^{-\ell-1})\zeta \leq e(\zeta) \leq C\zeta.$$

Then, for all $z \leq C$ we have

$$\Pr_{\mathrm{Z} \in \{0,1,\ldots,2^{\ell}-1\}} \left[z - e(\mathrm{Z} \cdot 2^{-2t}) \text{ is } (t, 2^{-3t-2})\text{-dangerous} \right] \leq 2^{-\ell+1}.$$

Proof. First, observe that if $z < 2^{-t-1}$ then $z - e(\mathrm{Z} \cdot 2^{-2t}) \leq z < 2^{-t-1}$ and therefore $z - e(\mathrm{Z} \cdot 2^{-2t})$ is not $(t, 2^{-3t-2})$-dangerous. Thus, we may assume that $z \geq 2^{-t-1}$. Using our assumption on e,

$$C(1 - 2^{-\ell-1})\mathrm{Z} \cdot 2^{-2t} \leq e(\mathrm{Z} \cdot 2^{-2t}) \leq C\mathrm{Z} \cdot 2^{-2t}.$$

Therefore,

$$\begin{aligned} e((\mathrm{Z}+1)2^{-2t}) - e(\mathrm{Z} \cdot 2^{-2t}) &\geq C(1 - 2^{-\ell-1})(\mathrm{Z}+1)2^{-2t} - C\mathrm{Z} \cdot 2^{-2t} \\ &= C2^{-2t}(1 - 2^{-\ell-1}(\mathrm{Z}+1)) \geq C2^{-2t-1} \geq 2^{-3t-2}, \end{aligned}$$

where we used that $C \geq z \geq 2^{-t-1}$ and $\mathrm{Z} \leq 2^{\ell} - 1$. In other words, $e(\mathrm{Z} \cdot 2^{-2t})$ is increasing with Z and $e(\mathrm{Z} \cdot 2^{-2t})$-s for consecutive Z-s are at least 2^{-3t-2} apart. Moreover,

$$e((2^{\ell} - 1)2^{-2t}) \leq (2^{\ell} - 1)2^{-2t}C < 2^{-t},$$

and so there are at most two Z-s for which $z - e(\mathrm{Z} \cdot 2^{-2t})$ is $(t, 2^{-3t-2})$-dangerous. □

Corollary 6.8. *Let $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$ be stochastic matrices of dimension $w \leq n$. Also, let $\ell, t \in \mathbb{N}$ be such that $t \geq 4 \log n$ and $\ell < t/2$. Then,*

$$\Pr_{\mathrm{Z} \in \{0,1,\ldots,2^{\ell}-1\}} \left[\prod_{i=1}^{n} (A_i - \mathrm{Z} \cdot 2^{-2t} J_w) \text{ is } (t, 2^{-3t-2})\text{-dangerous} \right] \leq w^2 2^{-\ell+1}.$$

Proof. For each entry (i,j), let $z = (A_1 \cdots A_n)[i,j]$ be the correct value of $A_1 \cdots A_n$ at entry (i,j). As before, let $E_{i,j}(\mathrm{Z} \cdot 2^{-2t})$ denote the noise introduced at entry (i,j) by the shift $\mathrm{Z} \cdot 2^{-2t}$. We know that

$$T[i,j] \cdot \mathrm{Z} \cdot 2^{-2t} \cdot (1 - wn\mathrm{Z} \cdot 2^{-2t}) \le E_{i,j}(\mathrm{Z} \cdot 2^{-2t}) \le T[i,j] \cdot \mathrm{Z} \cdot 2^{-2t}.$$

Set $C = T[i,j]$ and observe that

$$wn\mathrm{Z} \cdot 2^{-2t} < n^2 2^{\ell} 2^{-2t} \le 2^{t/2} 2^{\ell} 2^{-2t} \le 2^{-\ell-1},$$

and that $C < n^2 \le 2^{t-\ell}$. Furthermore,

$$z = (A_1 \cdots A_n)[i,j] \le T[i,j] = C.$$

Therefore, by Lemma 6.7, the probability over Z that

$$\prod_{i=1}^{n} (A_i - \mathrm{Z} \cdot 2^{-2t} J_w)[i,j]$$

is $(t, 2^{-3t-2})$-dangerous, is at most $2^{-\ell+1}$. The result then follows by the union bound over all w^2 entries. □

We union bound over all n sub-products, and get the following corollary.

Corollary 6.9. *Let $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$ be stochastic matrices of dimension $w \le n$. Also, let $\ell, t \in \mathbb{N}$, $\varepsilon \in (0,1)$ be such that $t \ge 4 \log n$ and $\ell < t/2$. Then, for the matrix sequence $M(\mathrm{Z}) = (A_1 - \mathrm{Z}2^{-2t} J_w, \ldots, A_n - \mathrm{Z}2^{-2t} J_w)$,*

$$\Pr_{\mathrm{Z} \in \{0,1,\ldots,2^{\ell}-1\}} \left[\exists i,j \ \prod_{L_{i,j}} (M(\mathrm{Z})) \text{ is } (t, 2^{-3t-2})\text{-dangerous} \right] \le nw^2 2^{-\ell+1}.$$

6.4 Proof of correctness

We recall our setting. Let (M) be a matrix sequence of substochastic matrices $(M)_1, \ldots, (M)_{2^r} \in \mathbb{R}^{w \times w}$. For a shift Z, the matrix sequence $(M_0(\mathrm{Z}))$ is defined by

$$(M_0(\mathrm{Z}))_i \triangleq (A_i - \mathrm{Z} \cdot 2^{-2t_2} J_w),$$

for $i = 1, \ldots, n$. We say Z is *good* if all sub-products $\prod_{L_{i,j}} (M_0(\mathrm{Z}))$ are $(t_2, 2^{-3t_2-2})$-safe. By Corollary 6.9, except for probability $nw^2 2^{-\ell+1}$, Z is good.

We now define two matrix sequences that represent evolution throughout the algorithm's iterations. The first matrix sequence (M_i) describes the situation in our algorithm if instead of applying $\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}$ we employ a true iterated product. Formally, it is recursively defined by

$$(M_i(\mathrm{Z}))_j \triangleq \mathsf{R}\left(\left\lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\rfloor_{t_2}, (M_{i-1}(\mathrm{Z}))_{\Gamma_{i,j}}, 8t_2\right). \qquad (14)$$

Notice that it only depends on Z (and not on h). We will eventually show that the matrix sequence (M_i) is very close to the matrix sequence $(\widetilde{M}_i)$. First, we claim that

Claim 6.10. *For all i,j,*

$$\left\| (M_i)_j - \prod_{L_{i,j}} (M_0) \right\|_\infty \le 2^{r_1 - 7t_2}.$$

We defer the proof for later. The second matrix sequence $(\widetilde{M}_i(\mathrm{Z},h))$ is defined by the behavior of the actual algorithm, i.e.,

$$(\widetilde{M}_i(\mathrm{Z},h))_j = \mathsf{R}\left(\left\lfloor \mathsf{R}\left(\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((\widetilde{M}_{i-1}(\mathrm{Z},h))_{\Gamma_{i,j}}, h, 2^{r_1}, t_1), (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 6t_2\right)\right\rfloor_{t_2}, (\widetilde{M}_{i-1}(\mathrm{Z},h))_{\Gamma_{i,j}}, 8t_2\right). \qquad (15)$$

This matrix sequence seemingly depends on both Z and h.

Now, say h is *good* for Z if h is good, in the sense of Theorem 6.2, for all sequences $(\widetilde{M}_{i-1}(\mathrm{Z},h))_{\Gamma_{i,j}}$. Namely, that

$$\left\| \Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((\widetilde{M}_{i-1}(\mathrm{Z},h))_{\Gamma_{i,j}}, h, 2^{r_1}, t_1) - \prod_{\Gamma_{i,j}} (M_{i-1}) \right\|_\infty \le \varepsilon_\Lambda + w2^{r_1} \cdot 2^{-t_1} \qquad (16)$$

for all $i \in [r_2]$ and $j \in [2^{r-r_1 i}]$. The main claim of this subsection is that when Z is good and h is good for Z, $(\widetilde{M}_i(\mathrm{Z},h))$ does not depend on h! Formally,

Lemma 6.11. *Suppose Z is good and h is good for Z. Then, for every $0 \le i \le r_2$,*

$$(\widetilde{M}_i(\mathrm{Z},h)) = (M_i(\mathrm{Z})).$$

Before proving Claim 6.10 and Lemma 6.11, we deduce the correctness of our algorithm.

Proof of Theorem 6.3. By Theorem 6.2, for every 2^{r_1} matrix sequence, h is good except for probability at most δ_Λ over h, and by the union bound, h is good for all these sequences except for probability at most $n\delta_\Lambda$. Thus, Z is good except for probability $nw^2 2^{-\ell+1}$ and h is good for Z except for probability $n\delta_\Lambda$. By Lemma 6.11 for a good Z and h, the sequences $(\widetilde{M}_i(\mathrm{Z},h))$ that the algorithm constructs are identical to the sequences $(M_i(\mathrm{Z}))$. By Claim 6.10,

$$\left\|(M_i(\mathrm{Z}))_j - \prod_{L_{i,j}}(M_0(\mathrm{Z}))\right\|_\infty \leq 2^{r_1-7t_2},$$

and as the entry-wise shifts are bounded by $2^{-3t_2/2}$,

$$\left\|\prod_{L_{i,j}}(M_0(\mathrm{Z})) - M_1\cdots M_n\right\|_\infty \leq w\cdot\left\|\prod_{L_{i,j}}(M_0(\mathrm{Z})) - M_1\cdots M_n\right\|_{\max} \leq nw2^{-3t_2/2}.$$

Recalling that $(\widetilde{M}_{r_2}(\mathrm{Z}))$ is the output of our algorithm, the above inequalities imply

$$\Pr_{\mathrm{Z},h}\left[\left\|\mathsf{SZ}_{\mathsf{IMM}}((M),h,\mathrm{Z}) - M_1\cdots M_n\right\|_\infty > 2^{r_1-7t_2} + nw2^{-3t_2/2}\right] \leq n(w^2 2^{-t_2/2+2} + \delta_\Lambda).$$

The parameters of Theorem 6.3 follow from our choice of parameters. □

Let us now proceed with the proofs of Claim 6.10 and Lemma 6.11.

Proof of Lemma 6.11. Fix any good Z and any h that is good for Z. For brevity, from now on we omit the dependence of $(M_i(\mathrm{Z}))$ on Z and write (M_i) instead, and we also omit the dependence of $(\widetilde{M}_i(\mathrm{Z},h))$ on Z and h and write $(\widetilde{M}_i)$ instead.

The proof is by induction on i. The base case, $i=0$ is trivial. Assume correctness for $i-1$, namely that $(\widetilde{M}_{i-1}(\mathrm{Z},h)) = (M_{i-1}(\mathrm{Z}))$, and let us prove for i. As h is good for the matrix sequences $(M_{i-1})_j$, see Equation (16), for every $i\in[r_2]$ and $j\in[2^{r-r_1 i}]$ we have that

$$\left\|\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((M_{i-1})_{\Gamma_{i,j}},h,2^{r_1},t_1) - \prod_{\Gamma_{i,j}}(M_{i-1})\right\|_\infty \leq \varepsilon_\Lambda + w2^{r_1-t_1} \leq \frac{1}{4(2^{r_1}+1)},$$

where we have used Equation (12). Therefore, by Lemma 3.9,

$$\left\|\mathsf{R}\Big(\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((M_{i-1})_{\Gamma_{i,j}},h,2^{r_1},t_1),(\widetilde{M}_{i-1})_{\Gamma_{i,j}},6t_2\Big) - \prod_{\Gamma_{i,j}}(M_{i-1})\right\|_\infty < 2^{r_1}2^{-6t_2} \leq 2^{-5t_2}, \tag{17}$$

using Equation (13). Let

$$A = \mathsf{R}\Big(\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((M_{i-1})_{\Gamma_{i,j}},h,2^{r_1},t_1),(\widetilde{M}_{i-1})_{\Gamma_{i,j}},6t_2\Big).$$

We know that A is close to $\prod_{\Gamma_{i,j}}(M_{i-1})$. Our strategy is to show that A is close to $\prod_{L_{i,j}}(M_0)$, which is a safe matrix (because Z is good) hence A would also be safe. Indeed,

$$\begin{aligned}\left\| \prod_{\Gamma_{i,j}}(M_{i-1}) - \prod_{L_{i,j}}(M_0) \right\|_\infty &= \left\| \prod_{k\in\Gamma_{i,j}} (M_{i-1})_k - \prod_{k\in\Gamma_{i,j}} \prod_{L_{i-1,k}} (M_0) \right\|_\infty \\ &\le \sum_{k\in\Gamma_{i,j}} \left\| (M_{i-1})_k - \prod_{L_{i-1,k}} (M_0) \right\|_\infty \le 2^{r_1}\cdot 2^{r_1-7t_2} \le 2^{-6t_2}.\end{aligned}$$

The first equality is Equation (11), and the following inequality is due to Claim 3.2. Finally, we used Claim 6.10 and that $r_1 \le t_2$. Thus, by the triangle inequality, A is $2^{-5t_2} + 2^{-6t_2}$ close to $\prod_{L_{i,j}}(M_0)$.

We now recall that since Z is good we have that $\prod_{L_{i,j}}(M_0)$ is $(t_2, 2^{-3t_2-2})$-safe. Thus, by definition, $\prod_{L_{i,j}}(M_0)$ is at least 2^{-3t_2-2}-far from a 2^{-t_2} boundary. As

$$2^{-5t_2} + 2^{-6t_2} < 2^{-3t_2-2},$$

for a good h, both A and $\prod_{L_{i,j}}(M_0)$ belong to the same 2^{-t_2} segment, i.e.,

$$\left\lfloor \prod_{\Gamma_{i,j}}(M_{i-1}) \right\rfloor_{t_2} = \lfloor A \rfloor_{t_2}.$$

As

$$\begin{aligned}(M_i)_j &= \mathsf{R}\left(\left\lfloor \prod_{\Gamma_{i,j}}(M_{i-1}) \right\rfloor_{t_2}, (M_{i-1})_{\Gamma_{i,j}}, 8t_2 \right), \\ (\widetilde{M_i})_j &= \mathsf{R}\left(\lfloor A \rfloor_{t_2}, (M_{i-1})_{\Gamma_{i,j}}, 8t_2 \right),\end{aligned}$$

we see that $(M_i)_j = (\widetilde{M_i})_j$, as desired. In particular, all good h-s give exactly the same matrix sequence (M_i). $\square$

Proof of Claim 6.10. We recall the setting. (M) is a matrix sequence of sub-stochastic matrices $(M)_1, \ldots, (M)_{2^r} \in \mathbb{R}^{w\times w}$. For a shift Z the matrix sequence $(M_0(\mathrm{Z}))$ is defined by

$$(M_0(\mathrm{Z}))_i = (A_i - \mathrm{Z}\cdot 2^{-2t_2} J_w),$$

for $i = 1, \ldots, n$. We define a matrix sequence (M_i) that describes the situation in our algorithm if instead of applying $\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}$ we employ a true iterated product. It is recursively defined by

$$(M_i(\mathrm{Z}))_j = \mathsf{R}\left(\left\lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\rfloor_{t_2}, (M_{i-1}(\mathrm{Z}))_{\Gamma_{i,j}}, 8t_2\right).$$

Notice that, indeed, the matrices only depend on Z (and not on h). We turn to prove by induction on i that

$$\left\| (M_i(\mathrm{Z}))_j - \prod_{L_{i,j}} (M_0) \right\|_\infty \le \varepsilon_i \triangleq 2^{r_1 - 8t_2} \cdot \left(2^{ir_1} - 1\right) \tag{18}$$

for all $j = 1, \dots, \frac{n}{2^{ir_1}}$.

The base case $i = 0$ is trivial. By induction, $\left\| (M_{i-1}(\mathrm{Z}))_k - \prod_{L_{i-1,k}} (M_0) \right\|_\infty \le \varepsilon_{i-1}$ for all $k = 1, \dots, \frac{n}{2^{(i-1)r_1}}$. Note that

$$\left\| \left\lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\rfloor_{t_2} - \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\|_\infty \le w2^{-t_2} \le \frac{1}{4(2^{r_1}+1)},$$

using Equation (13). By the Richardson iteration, Lemma 3.9,

$$\left\| (M_i(\mathrm{Z}))_j - \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\|_\infty = \\ \left\| \mathsf{R}\left(\left\lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\rfloor_{t_2}, (M_{i-1}(\mathrm{Z}))_{\Gamma_{i,j}}, 8t_2\right) - \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) \right\|_\infty \le (2^{r_1}+1)2^{-8t_2}.$$

Also, by Equation (11) and by the induction hypothesis,

$$\left\| \prod_{\Gamma_{i,j}} (M_{i-1}(\mathrm{Z})) - \prod_{k \in \Gamma_{i,j}} \prod_{L_{i-1,k}} (M_0) \right\|_\infty \le 2^{r_1} \cdot \varepsilon_{i-1}$$

for all $j = 1, \dots, \frac{n}{2^{ir_1}}$. The same bound also holds if we consider only a sub-sequence of

$\Gamma_{i,j}$. Putting it altogether, we get

$$\left\| (M_i(\mathrm{Z}))_j - \prod_{L_{i,j}} (M_0) \right\|_\infty \leq (2^{r_1}+1)2^{-8t_2} + 2^{r_1-8t_2} \cdot 2^{r_1} \cdot (2^{r_1(i-1)} - 1)$$
$$\leq 2^{r_1-8t_2}(2^{r_1 i} - 1) = \varepsilon_i.$$

This completes the induction. As $ir_1 \leq r_2 r_1 = \log n \leq t_2$, $\varepsilon_i \leq 2^{r_1-8t_2}2^{ir_1} \leq 2^{r_1-7t_2}$, as desired. □

6.5 The space complexity

We now analyze the space complexity of the algorithm. To start, let us try to employ the same approach as in Section 5. Define the functions f_i that take a sequence (M) of length n and output the sequence

$$f_i((M))_j = \mathsf{R}\Big(\big\lfloor \mathsf{R}\big(\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((M)_{\Gamma_{i,j}}, h, 2^{r_1}, t_1), M, 6t_2\big)\big\rfloor_{t_2}, (M)_{\Gamma_{i,j}}, 8t_2\Big)$$

which is exactly the computation done in the i-th iteration in the $\mathsf{SZ}_{\mathsf{IMM}}$ algorithm. Clearly, the composition $f_{r_2} \circ \cdots \circ f_1((M_0))$ computes "row by row" the output of the iterated Saks–Zhou algorithm on an input (M). The problem is that each f_i requires at least $r - ir_1 = \log n - i\sqrt{\log n}$ space just for indexing as j runs from 1 to 2^{r-ir_1}, and this accumulates to $\Omega(\log^{3/2} n)$ space. However, there is a redundancy in the above approach: Each f_i maintains a global index to its location in the recursion tree, and to eliminate this redundancy we will globally store that index. Each node computes the local function that takes a sequence (B) of length 2^{r_1} and outputs

$$f_i((B)) = \mathsf{R}\left(\left\lfloor \mathsf{R}\left(\Lambda_{\varepsilon_\Lambda,\delta_\Lambda}((B), h, 2^{r_1}, t_1), \prod(B), 6t_2\right)\right\rfloor_{t_2}, (M)_{\Gamma_{i,j}}, 8t_2\right), \tag{19}$$

where one should think of B as $(M)_{\Gamma_{i,j}}$, and the output as $(M_i)_j$. The above function only maintains indices locally, namely it does not know its location within the recursion. Knowing the local indices for every level of the recursion in tandem with the global location suffices for the algorithm to operate. We now formulate this idea, which results in a generalized space-bounded composition, and then explain how to instantiate it in order to derive Lemma 6.4.

6.5.1 Improved space bounded composition

We have seen the space composition theorem, Claim 3.3, where the space complexity of the composition is the sum of the space complexities of each separate layer. However, sometimes the composition can be implemented in a cheaper way. A notable example for that is the proof for $\mathbf{NC}^1 \subseteq \mathbf{L}$. An $\mathbf{NC}^1$ circuit has depth $O(\log n)$ and elementary Boolean functions as gates. Applying Claim 3.3, we get a simulation in $O(\log^2 n)$ space, because each of the $O(\log n)$ layers requires $O(\log n)$ bits just for keeping an index to its input. Looking at the $\mathbf{NC}^1 \subseteq \mathbf{L}$ proof, one sees that the cheaper simulation maintains indexing in a *global* way, thus avoiding the need to pay for an index at each layer. This global indexing is made possible because the computation has a *tree* structure.

In this section we prove Lemma 6.13 which is a natural combination of the above two fundamental building blocks in space-bounded computation, i.e.,

- Composing space bounded functions; and
- Computing Boolean formulas in logarithmic space.

Definition 6.12. *Let G be a set of functions $g\colon \{0,1\}^\star \to \{0,1\}^\star$. A generalized Boolean formula F with gates in G over the input variables $x_1, \dots, x_n$ is a labeled directed acyclic graph as follows.*

- *Variables: Vertices with no incoming edges are labeled with a variable from $\{x_1, \dots, x_n\}$;*
- *Gates: Vertices with incoming edges are labeled with a function $g \in G$;*
- *Output: Vertices with no outgoing edges are labeled by a distinct output from $y_i \in \{y_1, \dots, y_\ell\}$; and*
- *Degree: All vertices have out-degree exactly 1, except for the output vertices which have out-degree 0.*

The Boolean formula computes the function $F\colon (\{0,1\}^\star)^n \to (\{0,1\}^\star)^\ell$,

$$F(x_1, \dots, x_n) = (y_1, \dots, y_\ell),$$

by placing the variables $x_1, \dots, x_n$ at their corresponding vertices in the graph and propagating the Boolean values to the output vertices.

Two spacial cases to bear in mind are:

- The case where the graph has $O(\log n)$ depth and G is the set of elementary Boolean functions (And, Or, Negation), which corresponds to an $\mathbf{NC}^1$ computation, and,

- The case where the graph is a directed path and G is the set of functions in $\mathbf{L}$, which corresponds to a general composition of logspace functions.

Our generalized space bounded composition statement goes as follows.

Lemma 6.13. *Suppose $f(x_1,\ldots,x_n)$ can be computed by a generalized formula with gates in G. Furthermore,*

- *Let S_0 denote the space complexity required to output the labeled graph with its labels;*
- *Let $s_v(m)$ denote the space complexity required for computing $g_v \in G$ on inputs of length m, where g_v is the function associated with v; and*
- *Let $\ell(v)$ denote an upper bound on the output length of g_v.*

Then, we can compute the function f in

$$O\left(S_0 + \max_{(v_1,\ldots,v_t)\in\mathcal{P}} \sum_{i=1}^{t} (s_{v_i}(\ell(v_i)) + \log \ell(v_i))\right)$$

space, where $\mathcal{P}$ is the set of all possible paths in the generalized formula.

For example, for $\mathbf{NC}^1$, $S_0 = O(\log n)$ and $s(v_i) = \ell(v_i) = O(1)$. In addition, it is possible to apply Lemma 6.13 for arbitrary acyclic graphs (that is, circuits rather than formulas) by first converting them into a formula, paying an additive term of $O(\log |\mathcal{P}|)$ in the space complexity. The proof is essentially a combination of composing space bounded functions and computing Boolean formulas space efficiently, so we only highlight needed modifications.[10]

Proof Sketch. We traverse the graph bottom-up, namely starting from the output gates, recursively computing the values of the children as follows.

1. Globally maintain the current node within the formula. This is stored at the beginning of the work-tape.
2. As in the space composition algorithm (Claim 3.3), we maintain a stack for every level of the recursion.

[10]The lemma uses basic principles that are well-known and used throughout the literature. In particular, the approach bears some similarities to the global-tape oracle machines outlined in Goldreich's book [Gol08]. Nonetheless, because it is an integral and subtle part of the argument, we decided to give the full proof.

3. Each stack maintains a "local" index to its input (which takes $\log \ell(v_i)$ space), along with a work-tape for the computation of the relevant function (which takes $s_{v_i}(\ell(v_i))$ space).

4. Whenever a specific value of a gate is needed, there are two options:

   i. If this is an input gate, we read it from the input-tape using the global index.

   ii. If this is not an input gate, then we open another (lower) level of recursion for computing it. In that case, we update the global location of the algorithm within the recursion.

□

6.5.2 Proof of Lemma 6.4

Our algorithm takes the form of a tree of depth r_2. The j-th node in the i-th layer corresponds to the matrix $(\widetilde{M_i})_j$, and is computed from its children in the tree, i.e.,

$$(\widetilde{M_i})_j = f(A_1, \ldots, A_{2^{r_1}})$$

for some suitable f as described above. Using Lemma 6.13, the algorithm can be implemented in space $O(S \cdot r_2)$, where S is an upper bound on the space complexity needed to computed each f. More specifically, recall that our algorithm takes the form of a tree with arity 2^{r_1}, depth r_2, and the local functions being f_i as defined in Equation (19). Using the notation of Lemma 6.13, the gates at depth i are labeled f_i, and the layout of this tree can be computed in space $O(\log n)$. Each local function f_i can be computed in space

$$O\left(\log\log n \cdot (\log w + r_1)\right).$$

This follows using the standard space composition theorem, Claim 3.3, applied with the space complexity guarantees of Theorem 6.2 and Lemma 3.8. Putting it together, and recalling that $r_1 = r_2 = \sqrt{\log n}$ we get that the overall space complexity of our $\mathsf{SZ}_{\mathsf{IMM}}$ algorithm is

$$O\left(\log\log n \cdot \left(\sqrt{\log n} \cdot \log w + \log n\right)\right).$$

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 2009.

[AKM$^+$20] 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 Symposium on Foundations of Computer Science (FOCS)*, pages 1295–1306. IEEE, 2020.

[Arm98] Roy Armoni. On the derandomization of space-bounded computations. In *Randomization and approximation techniques in computer science*, volume 1518 of *LNCS*, pages 47–59. Springer, 1998.

[BCG18] Mark Braverman, Gil Cohen, and Sumegha Garg. Hitting sets with near-optimal error for read-once branching programs. In *Proceedings of the 50th Annual Symposium on Theory of Computing (STOC)*, pages 353–362. ACM, 2018.

[BCP83] 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.

[Ber84] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information processing letters*, 18(3):147–150, 1984.

[BGG93] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *computational complexity*, 3(4):319–354, 1993.

[BHPP22] Andrej Bogdanov, William M. Hoza, Gautam Prakriya, and Edward Pyne. Hitting sets for regular branching programs. In *Proceedings of the 37th Computational Complexity Conference (CCC)*, pages 3:1–3:22. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[BRRY14] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. Pseudorandom generators for regular branching programs. *SIAM Journal on Computing*, 43(3):973–986, 2014.

[CDR$^+$21] Gil Cohen, Dean Doron, Oren Renard, Ori Sberlo, and Amnon Ta-Shma. Error reduction for weighted PRGs against read once branching programs. In *Proceedings of the 36th Computational Complexity Conference (CCC)*, pages 22:1–22:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[CDS22] Gil Cohen, Dean Doron, and Ori Sberlo. Approximating large powers of stochastic matrices in small space. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 8, 2022.

[CH20] Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 10:1–10:25. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[CL20a] Eshan Chattopadhyay and Xin Li. Non-malleable codes, extractors and secret sharing for interleaved tampering and composition of tampering. In *Theory of Cryptography Conference*, pages 584–613. Springer, 2020.

[CL20b] Eshan Chattopadhyay and Jyun-Jie Liao. Optimal error pseudodistributions for read-once branching programs. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 25:1–25:27. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[De11] Anindya De. Pseudorandomness for permutation and regular branching programs. In *Proceedings of the 26th Computational Complexity Conference (CCC)*, pages 221–231. IEEE, 2011.

[DMR$^+$21] Dean Doron, Raghu Meka, Omer Reingold, Avishay Tal, and Salil Vadhan. Pseudorandom generators for read-once monotone branching programs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 58:1–58:21. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[Ebe89] Wayne Eberly. Very fast parallel polynomial arithmetic. *SIAM Journal on Computing*, 18(5):955–976, 1989.

[FK18] Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In *Proceedings of 59th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 946–955. IEEE, 2018.

[Gil77] John Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[Gol11] Oded Goldreich. A sample of samplers: a computational perspective on sampling. In *Studies in complexity and cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2011.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *Journal of the ACM*, 56(4):20, 2009.

[GW97] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Struct. Algorithms*, 11(4):315–343, 1997.

[HAB14] William Hesse, Eric Allender, and David A. Mix Barrington. Corrigendum to "Uniform constant-depth threshold circuits for division and iterated multiplication" [J. Comput. System Sci. 65 (4) (2002) 695–716]. *Journal of Computer and System Sciences*, 80(2):496–497, 2014.

[HK18] William M. Hoza and Adam R. Klivans. Preserving randomness for adaptive algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 43:1–43:19. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2018.

[Hoz21] William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 28:1–28:23. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[Hoz22] William Hoza. Recent progress on derandomizing space-bounded computation. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 121, 2022.

[HPV21] William M. Hoza, Edward Pyne, and Salil Vadhan. Pseudorandom generators for unbounded-width permutation branching programs. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 7:1–7:20. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[HU21] William M. Hoza and Chris Umans. Targeted pseudorandom generators, simulation advice generators, and derandomizing logspace. *SIAM Journal on Computing*, pages STOC17–281, 2021.

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Proceedings of the 23th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 672–683. Springer, 2006.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual Symposium on Theory of Computing (STOC)*, pages 356–364. ACM, 1994.

[Jun81] H. Jung. Relationships between probabilistic and deterministic tape complexity. In *Mathematical Foundations of Computer Science (MFCS)*, volume 118 of *LNCS*, pages 339–346. Springer, 1981.

[KNP11] Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. Pseudorandom generators for group products. In *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC)*, pages 263–272. ACM, 2011.

[KT22] Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 12:1–12:14. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[MP00] Carlo Mereghetti and Beatrice Palano. Threshold circuits for iterated matrix product and powering. *RAIRO-Theoretical Informatics and Applications*, 34(1):39–46, 2000.

[MRT19] Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. In *Proceedings of the 51st Annual Symposium on Theory of Computing (STOC)*, pages 626–637. ACM, 2019.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis94] Noam Nisan. $\mathbf{RL} \subseteq \mathbf{SC}$. *computational complexity*, 4(1):1–11, 1994.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[PV21] Edward Pyne and Salil Vadhan. Pseudodistributions that beat all pseudorandom generators. In *Proceedings of the 36th Computational Complexity Conference (CCC)*, pages 33:1–33:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[PV22] Edward Pyne and Salil Vadhan. Deterministic approximation of random walks via queries in graphs of unbounded size. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 57–67. SIAM, 2022.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In *31st Annual Symposium on Theory of Computing (STOC 1999)*, pages 159–168. ACM, 1999.

[RSV13] Omer Reingold, Thomas Steinke, and Salil Vadhan. Pseudorandomness for regular branching programs via Fourier analysis. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 8096 of *LNCS*, pages 655–670. Springer, 2013.

[RT92] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM Journal on Computing*, 21(5):896–908, 1992.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, pages 157–187, 2002.

[Sak96] Michael Saks. Randomization and derandomization in space-bounded computation. In *Proceedings of the 11th Computational Complexity Conference (CCC)*, pages 128–149. IEEE, 1996.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sim81] Janos Simon. On tape-bounded probabilistic Turing machine acceptors. *Theoretical Computer Science*, 16(1):75–91, 1981.

[Ste12] Thomas Steinke. Pseudorandomness for permutation branching programs without the group theory. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 19, page 83, 2012.

[SVW17] Thomas Steinke, Salil Vadhan, and Andrew Wan. Pseudorandomness and Fourier-growth bounds for width-3 branching programs. *Theory of Computing*, 13(1):1–50, 2017.

[SZ99] Michael E. Saks and Shiyu Zhou. $\mathbf{BP_HSPACE}(\mathrm{S}) \subseteq \mathbf{DSPACE}(\mathrm{S}^{2/3})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[Ta-13] Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *Proceedings of the 45th Annual Symposium on Theory of Computing (STOC)*, pages 881–890. ACM, 2013.

[Vad12] Salil Vadhan. *Pseudorandomness*. Now, 2012.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.

A Spectral Algorithm for Matrix Powering

In this section we prove Theorem 1.2. The idea is to use the Cayley-Hamilton Theorem as was done, e.g., in [MP00]. The algorithm for computing A^n given $A \in \mathbb{R}^{w \times w}$ is as follows.

1. Compute the characteristic polynomial of A and denote it by $p(X)$.
2. Compute $r(X) = X^n \bmod p(X)$, where $\deg(r) < \deg(p) = w$.
3. Compute $r(A)$.

To implement the above in a space-efficient manner, we use the following two results from parallel computation. The first is due to Berkowitz, who gave a parallel algorithm for computing the characteristic polynomial.

Theorem A.1 ([Ber84]). *There exists a logspace uniform family of* $\mathbf{NC}^2$ *circuits that computes the characteristic polynomial of a given matrix. In terms of space complexity, on input* $A \in \mathbb{R}^{w \times w}$ *the algorithm runs in space* $O(\log |A| \cdot \log w)$.

The second algorithm is for polynomial division.

Theorem A.2 ([Ebe89]). *Division of polynomials over the integers can be done in logspace uniform* $\mathbf{NC}^1$.

It turns out that one can even perform polynomial division, and various other polynomial (and integer) arithmetic in $\mathbf{TC}^0$ (see, e.g., [RT92] and references therein).

Claim A.3. *The above algorithm to compute* A^n *can be implemented in space* $O(\log n + \log w \cdot \log |A|)$.

Proof. By Theorem A.1, Item 1 can be done in $O(\log w \cdot \log |A|)$ space. Item 2, following Theorem A.2, can be done in $O(\log(n \cdot |A| w^2))$ space, and Item 3 can be done in $O(\log^2 w +$

$\log w \cdot \log |A|)$ space using Claim 3.6. The overall space complexity then follows from composition of space-bounded functions.

The correctness of the algorithm follows from the Cayley–Hamilton Theorem which states that if $p(X)$ is the characteristic polynomial of a matrix A then $p(A) = 0$. Since $r(X) = X^n \bmod p(X)$ there exists a polynomial $q(X)$ such that $X^n = q(X)p(X) + r(X)$ and so

$$A^n = q(A)p(A) + r(A) = r(A).$$

□

B Richardson iteration

In this section, for completeness, we prove Lemma 3.9.

Proof of Lemma 3.9. The algorithm constructs the following pair of block matrices which consists of $(n+1) \times (n+1)$ blocks of $w \times w$ matrices. For $0 \le i, j \le n$,

$$\mathcal{A}[i,j] = \begin{cases} -A & j = i+1, \\ I_w & i = j, \\ 0 & \text{otherwise}. \end{cases} \qquad \mathcal{B}[i,j] = \begin{cases} B_{i,j-1} & i < j, \\ I_w & i = j, \\ 0 & i > j. \end{cases}$$

The algorithm then outputs the matrix $\mathsf{R}(\mathcal{A}, \mathcal{B}, k)[1, n+1]$ as given in Section 3.4.

The space complexity of the algorithm follows by Claim 3.6. As for the correctness, first observe that

$$\mathcal{A}^{-1}[i,j] = \begin{cases} A_i \cdots A_{j-1} & i < j, \\ I_w & i = j, \\ 0 & i > j. \end{cases}$$

By our assumption $\|\mathcal{A}^{-1}[i,j] - \mathcal{B}[i,j]\|_\infty \le \frac{1}{4(n+1)}$ for every $0 \le i, j \le n$, and so

$$\left\|\mathcal{A}^{-1} - \mathcal{B}\right\|_\infty \le \frac{1}{4}.$$

Lastly, note that $\|\mathcal{A}\|_\infty \le 2$ and by the sub-multiplicativity of $\|\cdot\|_\infty$ we get

$$\|I - \mathcal{B}\mathcal{A}\|_\infty \le \left\|(\mathcal{A}^{-1} - \mathcal{B})\mathcal{A}\right\|_\infty \le \left\|\mathcal{A}^{-1} - \mathcal{B}\right\|_\infty \cdot \|\mathcal{A}\|_\infty \le \frac{1}{2}.$$

The correctness now follows by Lemma 3.8.

□

C Improving the Dependence on the Confidence Parameter in Nisan's generator

In this section we give the proof of Theorem 6.2 and discuss related concepts. Let us start by giving a different perspective on the Nisan generator (See Section 3.5). Recall that the Nisan generator has two types of inputs: a symbol x, and a sequence of hash functions $h = (h_1, \ldots, h_k)$. Define the collection of functions

$$\left\{ G_h(x) \triangleq \mathsf{N}(x, h) : h \in \mathcal{H}^k \right\}. \tag{20}$$

Another way to interpret Theorem 3.10 is that for a predetermined BP, if we *sample* a function from the collection (20) then, with high probability, it fools that BP with accuracy ε_{N}. This special "sampling property" of the Nisan generator was used to derive two of the most important derandomization results of **BPL** [Nis94, SZ99].

In hindsight, the aforementioned special property of the Nisan generator can be obtained generically using a primitive called an *averaging sampler*. By "generically" we mean that there is a simple procedure that endows a given PRG with this "sampling property". Furthermore, this procedure, which by now known as "Armoni's sampler trick", allows us to get an improved version of the Nisan generator (or any PRG for that matter) with better dependence on the confidence parameter. Recall that we used the Nisan generator to devise a randomized algorithm for approximating the powers of stochastic matrices (Theorem 3.14). Precisely in the same fashion, we use the aforementioned improved version of the Nisan generator to derive Theorem 6.2. The idea of using samplers in the context of PRGs against the class of BPs dates back to the work of Armoni [Arm98] and was recently used in the context of space-bounded computation in several works (e.g., [BCG18, CL20a, Hoz21]).

We now introduce the notion of samplers, towards deriving Theorem 6.2. The presentation mainly follows the highly recommended survey [Gol11].

Definition C.1. *An (ε, δ) sampler, is an algorithm $\Gamma^f \colon \{0,1\}^m \to [0,1]$ with oracle access to a function $f \colon \{0,1\}^n \to [0,1]$ such that*

$$\forall f \quad \Pr_y \left[\left| \Gamma^f(y) - \mathop{\mathbb{E}}_{\sigma}[f(\sigma)] \right| \geq \varepsilon \right] \leq \delta.$$

We refer to ε and δ as the accuracy parameter and confidence parameter, respectively.

In our setting, the quality of a sampler is determined by three complexity measures:

i. Space Complexity: The space required to run the sampler.

ii. Query Complexity: The number of oracle calls to the function $f\colon \{0,1\}^n \to [0,1]$.

iii. Randomness Complexity: The amount of randomness used by the sampler. In the above definition, we denoted it by m.

A very important type of samplers is that of *averaging samplers*, in which the sampler simply outputs the average of its queries. An *averaging sampler* can thus be defined via $\Gamma\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^\ell$, where the output of $\Gamma^f(y)$ is

$$\Gamma^f(y) = \mathop{\mathbb{E}}_{z\in\{0,1\}^d}[f(\Gamma(y,z))],$$

and so the query complexity of the above sampler is 2^d.[11]

Let us defer the discussion on the existence of efficient samplers, and go back to pseudorandom generators. Suppose that we have a PRG $G\colon \{0,1\}^\ell \to \Sigma^n$ fooling $[n,w,\Sigma]$ BPs with error ε, i.e., for every $[n,w,\Sigma]$ BP B, $\|\mathbb{E}_x[B(G(x))] - \mathbb{E}_\sigma[B(\sigma)]\|_\infty \le \varepsilon$. Note that G does not necessarily satisfy the guarantee of the Nisan generator discussed above and can be any PRG. Setting the function $f = B \circ G$, it is natural to try and sample from f using Γ^f. It is worth pointing out that if Γ is an averaging sampler then G_Γ is defined by $G_\Gamma(y,z) = G(\Gamma(y,z))$. Using a non-averaging sampler, we would have obtained a randomized *algorithm* that approximates the transition matrix in a black-box fashion, and for non-adaptive samplers the approximation is "oblivious". (See [CH20] for another discussion on various notions of sampling in the context of space-bounded derandomization.)

As implied by the discussion above, the main advantage of using Γ^f rather than simply computing $\mathbb{E}[f]$ is the strong guarantee that with high probability, namely, with probability $1-\delta$ *over the fixing of the input y to the sampler*, the approximation is good. Typically, the sampler's space complexity is smaller than the seed length of G so we can get a space-efficient approximation with high probability. Using good enough samplers, we can guarantee a very small δ without paying for it in the parameters of G. Also, for a good Γ, the randomness complexity is comparable to the seed length of G. For completeness, we give the precise details in the following lemma.

Lemma C.2 (the sampler trick for IMM). *The following holds for any positive integers n, w and s, and any $\varepsilon, \delta > 0$. Let $G\colon \{0,1\}^\ell \to \Sigma^n$ be a PRG fooling $[n,w,\Sigma]$ BPs with error $\frac{\varepsilon}{2}$.*

[11] More generally, one can consider non-adaptive samplers in which $\Gamma^f(y) = g_y\left(\left(f(\Gamma(y,z)\right)_{z\in\{0,1\}^d}\right)$, where g_y are arbitrary functions.

Assume that given $x \in \{0,1\}^\ell$ and $i \in [n]$, the symbol $G(x)_i$ can be computed in space $s_G = s_G(n, w, |\Sigma|, \varepsilon)$. Also, let Γ be an $(\varepsilon_{\mathsf{samp}}, \delta_{\mathsf{samp}})$-sampler with randomness complexity ℓ' and space complexity s_{samp}, outputting ℓ bits, where $\varepsilon_{\mathsf{samp}} = \frac{\varepsilon}{2w}$ and $\delta_{\mathsf{samp}} = \frac{\delta}{w^2 n^2}$.

Then, there exists an algorithm Λ that gets as input a seed $h \in \{0,1\}^{\ell'}$ and substochastic matrices $(A_1, \ldots, A_n) \in \mathbb{R}^{w \times w}$ in which every entry is represented using $\log|\Sigma|$ bits, and outputs $(M_h)_{a,b} \in \mathbb{R}^{w \times w}$ for all $1 \le a < b \le n$ such that the following holds. With probability at least $1 - \delta$ over h, for all $1 \le a < b \le n$ it holds that

$$\|(M_h)_{a,b} - A_a \cdots A_b\|_\infty \le \varepsilon.$$

Λ runs in space $O(\log(nw \cdot \log|\Sigma|) + s_G + s_{\mathsf{samp}})$. Moreover, if $\Gamma \colon \{0,1\}^{\ell'} \times \{0,1\}^d \to \{0,1\}^\ell$ is an averaging sampler, Λ runs in

$$O\left(\log(nw \cdot \log|\Sigma|) + d + s_G + \bar{s}_{\mathsf{samp}}\right)$$

space, where $\bar{s}_{\mathsf{samp}}$ is the space it takes to compute each $\Gamma(y,z)$ on any given $y \in \{0,1\}^{\ell'}$ and $z \in \{0,1\}^d$.

Proof. Fix $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$. Consider the canonical $[n, w+1, \Sigma]$ BP B that simulates $A_1, \ldots, A_n$, in which every layer as described in Section 3.5. Namely, using the terminology of Section 3.5, the i-th layer of B is $\mathsf{C}(A_i)$, and it holds that $\mathsf{A}(\mathsf{C}(A_i))_{[1,w]} = A_i$.

Let $B_{(s,i)\to(t,j)}$ be the sub-BP of length $t - s$ whose start node is the i-th state in the s-th level and its accept node is the j-th state in the t-th level. Fix any $1 \le a < b \le n$ and $i \in [w]$. By our guarantee on G, it holds that

$$\sum_{j \in [w]} \left| (A_a \cdots A_b)[i,j] - \mathbb{E}_{x \in \{0,1\}^\ell}\left[B_{(a-1,i)\to(b,j)}(G(x))\right] \right| \le \frac{\varepsilon}{2}.[12]$$

Considering our sampler Γ, define the function

$$f_j^{a,b,i}(x) = (A_a \cdots A_b)[i,j] - \mathbb{E}_{x \in \{0,1\}^\ell}\left[B_{(a-1,i)\to(b,j)}(G(x))\right],$$

noting that $\sum_j \left|\mathbb{E}[f_j^{a,b,i}(x)]\right| \le \frac{\varepsilon}{2}$. For brevity, we omit the tuple (a,b,i). Define the set of bad sample points

$$\mathbf{B}_j = \left\{ h \in \{0,1\}^{\ell'} : \left|\Gamma^{f_j}(h) - \mathbb{E}[f_j]\right| > \varepsilon_{\mathsf{samp}} \right\}.[13]$$

[12]Here and throughout the proof, we truncate the output of G accordingly.
[13]Recall that if Γ is an averaging sampler, $\Gamma^{f_j}(h)$ corresponds to $\mathbb{E}_{z \in \Gamma(h)}[f_j(z)]$.

By the sampler property, we know that $\mu(\mathbf{B}_j) \le \delta_{\mathsf{samp}}$. Denoting $\mathbf{B} = \bigcup_j \mathbf{B}_j$, we get that $\mu(\mathbf{B}) \le w\delta_{\mathsf{samp}}$. For $h \notin \mathbf{B}$,

$$\sum_{j\in[w]} \left|\Gamma^{f_j}(h)\right| \le \sum_{j\in[w]} \left(|\mathbb{E}[f_j]| + \varepsilon_{\mathsf{samp}}\right) \le \frac{\varepsilon}{2} + w \cdot \varepsilon_{\mathsf{samp}} \le \varepsilon.$$

Then, for a seed h, the (a,b)-th output of Λ on index $[i,j]$ is computed as follows: Compute the corresponding sub-BP $\overline{B} = B_{(a-1,i)\to(b,j)}$, and output $\Gamma^{\overline{B}\circ G}(h)$. When Γ is an averaging sampler, this corresponds to taking the average of $\overline{B}(G(\Gamma(h,z)))$ over all $z \in \{0,1\}^d$. To establish correctness, observe that if $h \notin \mathbf{B}$ we get

$$\sum_{j\in[w]} \left|\Gamma^{\overline{B}\circ G}(y) - (A_a \cdots A_b)[i,j]\right| \le \sum_{j\in[w]} \left|\Gamma^{f_j}(h)\right| \le \varepsilon.$$

To conclude, recall that $\mathbf{B}$ was defined with respect to (a,b,i). Union-bounding over all such tuples, we get that our algorithm works with probability at least $1 - n^2 w \cdot w\delta_{\mathsf{samp}} = 1 - \delta$ over the h-s.

Finally, we establish the space complexity of outputting a single entry in a single output matrix. Computing $\overline{B}$ requires $O(\log\log|\Sigma| + \log w)$ space. By composition of space-boudned functions, the sampling takes $O(\log(nw) + s_G + s_{\mathsf{samp}})$ space. The "Moreover" part readily follows from noting that when Γ computes the average of 2^d samples, $s_{\mathsf{samp}} = O(d + \bar{s}_{\mathsf{samp}})$. □

In Appendix C.1 we give the following "median-of-averages" sampler.

Lemma C.3 (median-of-averages sampler [BGG93])**.** *For every integer m, and any $\varepsilon, \delta > 0$, there exists an (ε,δ) sampler Γ that queries functions from $\{0,1\}^m \to [0,1]$ with randomness complexity $n = m + O(\log\log n \cdot \log\frac{1}{\delta})$, space complexity $O(\log\frac{m}{\varepsilon} + \log\log\frac{1}{\delta})$, and query complexity 2^d for $d = O(\log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta})$.*

Instantiating Lemma C.2 with Nisan's PRG (Theorem 3.10) and the median-of-averages samplers, we establish the space-efficient Theorem 6.2 with the improved dependence on δ (that stems only from how the parameters of Γ depends on δ).[14]

As the name suggests, the median-of-averages sampler is not an averaging sampler and therefore the improved approximation of Theorem 6.2 is not based on a PRG. While not strictly necessary for our application of proving our result, for future applications it

[14]The algorithm in Theorem 6.2 gets a truncation parameter t as well, and incurs an additional $nw \cdot 2^{-t}$ additive factor in the approximation error. However, going from the $t = s$ case to $t < s$ is done exactly as in Theorem 3.14, and so we omit the details.

may be useful to have a space-efficient PRG with improved dependence on the confidence parameter δ. In Appendix C.2, we prove:

Lemma C.4 (follows from [RVW02, Gol11])**.** *For every integer m, and any $\varepsilon, \delta > 0$, there exists an (ε, δ) sampler $\Gamma\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that*

$$d = O\left(\log\log\frac{1}{\delta} + \log\frac{1}{\varepsilon} + \log\log m\right)$$

and $n = m + O(\log\frac{1}{\varepsilon\delta})$. Moreover, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, $\Gamma(x,y)$ can be computed in space $O(\log^2 m + \log m \cdot \log\frac{1}{\varepsilon})$.

Instantiating Lemma C.2 with Nisan's PRG and the above averaging sampler, we get parameter that are the same as in Theorem 6.2, up to doubly-logarithmic factors. We note however, that the non-averaging sampler of Lemma C.3 is significantly simpler so we choose to give the self-contained construction below.

While the constructions in Lemmas C.3 and C.4 are known, we need a tighther analysis of their space complexity than appears in the existing literature.

C.1 Using non-averaging samplers

We now describe the sampler of Lemma C.3, which has two ingredients:

1. A pairwise independent distribution over $(\{0,1\}^m)^t$ with $t = O(\frac{1}{\varepsilon^2})$. Assuming $t \le 2^m$, this can be implemented via taking linear combinations in $\mathbb{F}_{2^m}$, and can be sampled using $r = 2m$ bits (see, e.g., [Vad12, Section 3]). Note that arithmetic operations over $\mathbb{F}_{2^m}$ can be done in logarithmic space (see, e.g., [HAB14, HV06]).

2. An expander graphs over the vertex set $\{0,1\}^r$ with degree $D = \log n$ and a small enough constant spectral gap. Each vertex of the graph indexes an element of the pairwise independent distribution. We assume that random walks over that expander graph can be computed space-efficiently. That is, given the labels $\gamma_1, \dots, \gamma_\ell \in [D]^\ell$, and a starting vertex $v \in \{0,1\}^r$, we can compute the corresponding random walk in $O(\log \ell + \log\log D)$ space. For such graphs, see, e.g., [CDR+21, Appendix B].

Let $f\colon \{0,1\}^m \to [0,1]$ be any function. The sampler $\Gamma^f(y)$ then proceeds as follows.

- We use y to take a random walk of length $\ell - 1 = O(\log\frac{1}{\delta})$ on the expander graph. Note that indeed $|y| = n = m + \ell \cdot \log D$.

- We use the vertices along the path to obtain ℓ samples $(Z_1^{(j)}, \dots, Z_t^{(j)})_{j=1}^{\ell}$ of the pairwise independent distribution.

- We then take the average over every batch, $\mu_j = \mathbb{E}_i f(Z_i^{(j)})$, and output the median of those averages, $\mathrm{median}(\mu_1, \ldots, \mu_\ell)$.

A detailed analysis is provided in [Gol11]. The query complexity is $\ell \cdot t$, as stated. The space complexity follows from the above assertions, and by composition of space bounded algorithms (Claim 3.3).

Remark 2. Taking the median in Section 5.3 is in fact a naïve implementation of the median-of-averages sampler: it takes the median of independent samples, whereas the median-of-averages sampler takes the median of the dependent samples obtained by taking a random walk over an expander graph.

C.2 Using averaging samplers

The construction of Lemma C.4 essentially follows [RVW02, Gol11, CL20b]. To the best of our knowledge, the claim about the space complexity is new, and previous work that used Lemma C.4 only needed Γ to be computable in linear space. In order to attain good space complexity, we need to analyze various pseudorandomenss primitives, and some we only partially describe here.

We use standard notion from the extractors literature, all can be found, e.g., in [GUV09, Vad12]. Using the equivalence between averaging samplers and extractors [Zuc97], it suffices to prove the following.

Theorem C.5. *For every positive integer m, and any $\delta, \varepsilon > 0$, there exists a $(k = n - \Delta, \varepsilon)$ extractor $\mathsf{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $\Delta = \log \frac{1}{\delta} + 1$, such that $d = O(\log \Delta + \log \frac{1}{\varepsilon} + \log\log m)$ and $n = m + O(\Delta + \log \frac{1}{\varepsilon})$. Moreover, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, $\mathsf{Ext}(x, y)$ can be computed in space $O(\log^2 m + \log m \cdot \log \frac{1}{\varepsilon})$.*

We follow the proof in [CL20b, Appendix B], for which we will need two extractors. The first one is the GUV logarithmic-seed extractor [GUV09].

Theorem C.6 ([GUV09, KT22])**.** *For any positive integers n and $k \le n$, and any $\varepsilon > 0$ and a constant $\alpha > 0$, there exists a (k, ε) extractor $\mathsf{Ext}_1\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ where $d = O(\log(n/\varepsilon))$ and $m = (1-\alpha)k$. Moreover, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, $\mathsf{Ext}_1(x, y)$ can be computed in space $O(\log k \cdot \log n)$.*

The construction of Ext_1 in [GUV09] uses seeded condensers (in fact, constructing suitable condensers is the crux of the [GUV09] paper). When computing Ext_1 in linear space suffices, the condenser [GUV09] can be used. Here, for the simplicity of the analysis, we replace the GUV condenser with a recent construction of [KT22] since its (very small) space complexity follows quite easily. We will analyze the space complexity of Ext_1 below.

The second extractor is an extractor for high min-entropy with good dependence on the input entropy deficiency, given by Goldreich and Wigderson. We instantiate the extractor with the parameters given in [CDR$^+$21, Appendix B].

Theorem C.7 ([GW97])**.** *For any positive integers m and $\Delta < m$, and any $\varepsilon > 0$, there exists a $(k = m - \Delta, \varepsilon)$ extractor $\mathsf{Ext}_2\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^m$ where $d = O(\Delta + \log(m/\varepsilon))$. Moreover, given $x \in \{0,1\}^m$ and $y \in \{0,1\}^d$, $\mathsf{Ext}_2(x,y)$ can be computed in space $O(\log m \cdot \log d)$.*

In the case where the output length is the same as the input length, an application of Ext_2 simply amounts to taking a step on a expander over 2^m vertices. By using expanders with better dependence between the degree and the spectral gap, one can get $d = O(\Delta + \log \frac{1}{\varepsilon})$, as indeed [GW97] do get. However, the space complexity of the GW extractor as stated in existing literature is linear, i.e., $O(m + \log \frac{1}{\varepsilon})$, and we need better space complexity.[15] In [CDR$^+$21], we instantiate the GW extractor with a (sub-optimal) expander that comes from small-biased sets.

Proof of Theorem C.5. We instantiate our two extractors as follows.

- Let $\mathsf{Ext}_2\colon \{0,1\}^m \times \{0,1\}^{d_1} \to \{0,1\}^m$ be the $(k_1 = m - \Delta, \varepsilon/3)$ GW extractor from Theorem C.7. We have that $d_1 = O(\Delta + \log \frac{m}{\varepsilon})$, and we can assume that $d_1 \geq \Delta + \log \frac{3}{\varepsilon}$.
- Let $\mathsf{Ext}_1\colon \{0,1\}^{3d_1} \times \{0,1\}^d \to \{0,1\}^{d_1}$ be the $(2d_1, \varepsilon/3)$ extractor from Theorem C.6. Indeed, $d = O(\log(d_1/\varepsilon)) = O(\log \Delta + \log \frac{1}{\varepsilon} + \log\log m)$.

The extractor Ext is constructed as follows. Set $n = m + 3d_1$. Given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, write $x = (x_1, x_2) \in \{0,1\}^m \times \{0,1\}^{3d_1}$ and output

$$\mathsf{Ext}((x_1,x_2),y) = \mathsf{Ext}_2(x_1, \mathsf{Ext}_1(x_2,y)).$$

The correctness follows from "block-source extraction", and we repeat it here, briefly, for completeness. Given an (n,k) source X, recall that $H_\infty(X) \geq m + 3d_1 - \Delta$, and it is known (see, e.g., [GUV09, Lemma 4.15]) that X is close to a block source. Namely, it is $(\varepsilon/3)$-close to (X_1, X_2) in which $H_\infty(X_1) \geq m - \Delta$ and $H_\infty(X_2 | X_1 = x_1) \geq 3d_1 - \Delta - \log(3/\varepsilon)$ for every $x_1 \sim X_1$. For any fixing of $x_1 \sim X_1$, $\mathsf{Ext}_1(X_2, U_d)$ is $(\varepsilon/3)$-close to U_{d_1}, and one

[15]More precisely, we need to take a single step on a λ-expander with $\lambda \leq \frac{\varepsilon^2}{4 \cdot 2^{\Delta/2}}$, or alternatively, if we don't care about the exact constants, a $O(\Delta + \log \frac{1}{\varepsilon})$-step walk over a constant-gap expander. We do not know how to implement this with good constant-degree expanders and space complexity $\mathrm{polylog}(\Delta + \log \frac{1}{\varepsilon})$.

can show that in this case, $\mathsf{Ext}_2(X_1, \mathsf{Ext}_1(X_2, Y))$ is $(2\varepsilon/3)$-close to U_m (see, e.g., [GUV09, Lemma 4.13]). Incurring the distance from X to (X_1, X_2), we conclude that $\mathsf{Ext}(X, Y)$ is ε-close to U_m.

For the space complexity, note that Ext_2 can be computed in

$$O(\log m \cdot \log d_1) = O\left(\log m \cdot \left(\log \Delta + \log \frac{1}{\varepsilon} + \log\log m\right)\right)$$

space, and Ext_1 takes $O(\log k_1 \cdot \log m) = O(\log^2 m)$ space to compute. The theorem then follows from the composition of space-bounded algorithms (see Claim 3.3). $\square$

The GUV Extractor of Theorem C.6. Before we begin, we will need the following condenser, whose space complexity we analyze below.

Theorem C.8 ([GUV09, KT22]). *For any positive integers n and $k \le n$, and any $\varepsilon > 0$ and a constant $\alpha > 0$, there exists a $k \to_\varepsilon k + d$ condenser (i.e., a lossless condenser) $\mathsf{Cond}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that $d = O_\alpha(\log(n/\varepsilon))$ and $m = (1+\alpha)k$.*

Moreover, given $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, $\mathsf{Cond}(x, y)$ in [KT22] can be computed in space $O(\log n + \log\log \frac{1}{\varepsilon})$.

We note that the parameters we gave above are rather crude, but suffice for obtaining Theorem C.6. For the parameter regime of Theorem C.6, computing Ext_1 is done recursively as follows. For $t = O(\log k)$, we construct $\mathsf{E}_0, \dots, \mathsf{E}_t$ where $\mathsf{Ext}_1 = \mathsf{E}_t$.[16] The extractor E_0 makes a constant number of calls to the condenser of Theorem C.8 and a constant number of calls to an extractor based on the Leftover Hash Lemma (LHL). By invoking standard two-universal hash functions, an LHL-based extractor can be implemented in space $O(\log n)$.[17] Each extractor E_i makes one call to some $\mathsf{E}_{i'}$ for $i' < i$, a constant number of calls to a condenser, and a constant number of calls to an LHL extractor. We refer the reader to [GUV09] for the complete details.

Overall, using composition of space-bounded algorithms, the extractor of Theorem C.6 can be implemented in space $O(\log k \cdot \log n)$.[18]

The Space Complexity of Theorem C.8. Lastly, we wish to establish the space complexity of the condenser in Theorem C.8. We first give an overview of the construction and

[16]In fact, to increase the output length, one has to reiterate the construction using different parameters for a constant number of times. However, this does not change the stated space complexity bound.

[17]Although the parameters of each extractor invocation are different, we will always take the pessimistic bound of input length n and error $\frac{\varepsilon}{\mathrm{poly}(n)}$, which suffices.

[18]Note that we assume ε is at most $2^{-\Theta(n)}$, which we can do without loss of generality since we chose to not output the seed.

parameters of [KT22]. The authors set a prime field $\mathbb{F}_q$ for $q = \mathrm{poly}_\alpha(n/\varepsilon)$. Also, set $\overline{m} \in \mathbb{N}$ such that $2^m = q^{\overline{m}+2}$, and identify each seed in $\{0,1\}^d$ with an element of $\mathbb{F}_q$.[19] The parameters are set so that we can identify any $f \in \{0,1\}^n$ as a univariate polynomial over $\mathbb{F}_q$ with degree at most $\overline{n} - 1$. The output $\Gamma(f, y)$ is given by

$$\Gamma(f,y) = \big(y, f(y), f'(y), \ldots, f^{(\overline{m}+1)}(y)\big) \in \mathbb{F}_q^{\overline{m}+2}.$$

where $f^{(i)}$ is the (formal) i-th derivative of f. Fix some $i \in [\overline{m}+1]$, and consider the computation of some $f^{(i)}(y)$. It is known that iterated addition and multiplication can be done by logspace-uniform (and even logtime uniform) $\mathbf{TC}^0$ circuits, so in particular in $\mathbf{L}$. More concretely, adding and multiplying t $\mathbb{F}_q$-elements can be done by $\mathrm{poly}(t \log q)$-sized $\mathbf{TC}^0$ circuits (see, e.g., [HAB14]).[20] Computing $f^{(i)}(y)$ amounts to:

1. Computing $f^{(i)}$: Each coefficient of $f^{(i)}$ is a multiplication of at most $\overline{m}+2$ field elements, which can be computed in space $O(\log \overline{m} + \log\log q)$.

2. Evaluating $f^{(i)}(y)$: Amounts to computing exponentiation up to an $\overline{n}$-th power, and performing addition of up to $\overline{n}$ elements. This can be done in space $O(\log \overline{n} + \log\log q)$.

By composition of space bounded algorithms, the overall space requirement of computing the $\overline{m}$ field elements of $\Gamma(f, y)$ is

$$O(\log \overline{n} + \log\log q + \log \overline{m}) = O\left(\log n + \log\log \frac{1}{\varepsilon}\right).$$

[19]The fact that q is not a power of 2 can be addressed with a small loss in parameters.

[20]Computing a field element to the t-th power can even be done in size $\mathrm{poly}(\log t, \log q)$ for specific realizations of $\mathbb{F}_q$, see [HV06], but we won't need this fact.