Approximating Iterated Multiplication of Stochastic Matrices in Small Space

Gil Cohen
gil@tauex.tau.ac.il
Tel Aviv University
Tel Aviv, Israel

Dean Doron
deand@bgu.ac.il
Ben Gurion University of the Negev
Be'er Sheva, Israel

Ori Sberlo
orisberlo@mail.tau.ac.il
Tel Aviv University
Tel Aviv, Israel

Amnon Ta-Shma
amnon@tauex.tau.ac.il
Tel Aviv University
Tel Aviv, Israel

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 [32] 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 [32] was achieved by Hoza [15] 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 [32]. 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 framework of [32] 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.

CCS CONCEPTS

• **Theory of computation → Pseudorandomness and derandomization**; *Complexity classes*; Random walks and Markov chains.

KEYWORDS

space-bounded computation, derandomization, matrix powering, iterated matrix multiplication

ACM Reference Format:
Gil Cohen, Dean Doron, Ori Sberlo, and Amnon Ta-Shma. 2023. Approximating Iterated Multiplication of Stochastic Matrices in Small Space. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3564246.3585181

1 INTRODUCTION

One of the great open problems of computational complexity is the **BPL** vs. **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 **BPL** = **L** (as indeed follows from plausible circuit lower bounds [21]), and there are no known barriers for the unconditional derandomization of **BPL**.

The problem of derandomizing **BPL** is equivalent to the problem of approximating powers of stochastic matrices.[1] 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) **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., [19, 24, 25]). 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 [26] 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 [2] who considered a different range of parameters, and the work of Raz and Reingold [29] that put forth an approach for significantly improving upon [26]. Most relevant to us is the work of Saks and Zhou [32] which builds on Nisan's work [24]. Interestingly,

[1]Indeed, a Turing machine M that uses space $S = \log w$ on inputs of length m can be converted to a Markov chain A on $O(w \cdot m)$ states, and a Markov chain on Q states can be simulated in space $O(\log Q)$.

a recent line of work [4, 17, 27, 28] 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 [11, 30, 36]. In particular, Meka, Reingold and Tal [23] 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., [6, 9, 10, 22, 35] and references therein).

The $n \gg w$ regime: the non black-box model. The focus of this work is the latter regime, $n \gg w$, in the *non black-box* (or, 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 [33] 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 Theorem 3.2). 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 [32] runs in space

$$O\left(\sqrt{\log n} \cdot \log \frac{nw}{\varepsilon}\right).[2]$$

The dependence on ε was recently improved by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford, and Vadhan [1] using the Richardson iteration (see subsection 3.3), reducing the space to

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

Hoza [15] 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 [14] 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 [32] (and over [1]) 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 observe 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 [8, 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 [14], the early works on randomized space-bounded algorithms showed more interest in the "non-halting" model (see, e.g., [5, 12, 20, 31, 34]).

[2] [32] considers matrix powering. However, one can reduce IMM to matrix powering via the embedding $(A_1, \ldots, A_n) \mapsto \bar{A} = \begin{pmatrix} 0 & A_1 & & \\ & 0 & \ddots & \\ & & \ddots & \ddots \\ A_n & & & 0 \end{pmatrix}$. 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, $n \gg w$.

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 [32] framework and devise an intricate precision-alternating recursive scheme. We elaborate on this in subsection 2.1. The general case of iterated matrix multiplication poses additional significant challenges, the most 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 subsection 2.2.

In this conference version, we omit all proofs and skip most of the technical content. The full version of the paper, which we will often refer to, can be found in [8].

2.1 Matrix Powering

2.1.1 The [32] 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 [24], 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 [32] 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 $i = 1, \dots, \sqrt{\log n}$,
  (a) Invoke the Nisan generator to approximate $(\widetilde{A}_{i-1})^{2^{\sqrt{\log n}}}$ to within 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 $\{0, 1, \dots, L\}$ 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 satisfy 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 [32] 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 subsection 3.4) allows [32] to *reuse* the randomness needed for the different applications of the Nisan generator. This reuse of randomness saves on space in the resulting 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.[3]

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 [32] 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.

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

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 [1, 7, 27]. We turn to describe this algorithm.

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 $\|A^i - \widetilde{A}_i\|_\infty \le \frac{1}{n}$. The output, R, is a $w \times w$ matrix satisfying $\|A^n - R\|_\infty \le n \cdot 2^{-k}$, computed in space $O\left(\log^2 k + \log k \cdot \log nw\right)$. 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 subsection 3.3 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, zigzagging 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 [32], 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 [1] 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 $A_1, \ldots, A_n$, 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

$$\begin{tikzcd}
\widetilde{M}_0 \arrow[r, "\text{Richardson}"] \arrow[d, "\text{Truncate}"'] & \mathsf{R}_0 \arrow[r, "\text{S\&T}"'] & \widetilde{M}_1 \arrow[r, "\text{Richardson}"] \arrow[d, "\text{Truncate}"'] & \mathsf{R}_1 \arrow[r, "\text{S\&T}"'] & \widetilde{M}_2 \\
\lfloor \widetilde{M}_0 \rfloor \arrow[r, "\text{Nisan}"'] & \approx \lfloor \widetilde{M}_0 \rfloor^{2^{\sqrt{\log n}}} \arrow[u, "\text{Richardson}"] & \lfloor \widetilde{M}_1 \rfloor \arrow[r, "\text{Nisan}"'] & \approx \lfloor \widetilde{M}_1 \rfloor^{2^{\sqrt{\log n}}} \arrow[u, "\text{Richardson}"] & \cdots
\end{tikzcd}$$

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

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 subsection 6.2 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 an 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 [8, Section 6.5.1] for the details, and in particular [8, 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., subsection 3.4 or Theorem 3.8) and still be certain that with probability $1 - \delta$ over the auxiliary 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 Nisan's 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 subsection 6.1 for more details).

3 PRELIMINARIES

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.,

$$\|A\|_\infty = \max_{i\in[w]} \sum_{j\in[w]} |A[i,j]|\,.$$

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 nonnegative and every row sums to at most 1, i.e., $\|A\|_\infty \le 1$.

3.1 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 [3, Chapter 4] and [13, Chapter 5].

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 3.1 (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 $|A| = \Omega(w^2)$.

Using space-efficient composition of space-bounded functions, we can deduce:

Lemma 3.2. *The matrix powering function $f(A, n) = A^n$ can be computed in space $O(\log^2 n + \log n \cdot \log|A|)$.*

We leave the details to the full version.

3.2 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 $V_{\mathrm{acc}} \subseteq [w]$ and a sequence of transition functions $B_i : [w] \times \Sigma \to [w]$ for $i \in [n]$. The BP B naturally defines a function $B : \Sigma^n \to \{0, 1\}$: Start at v_0, and then for $i = 1, \dots, n$ 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 $v_n \in V_{\mathrm{acc}}$, and rejects otherwise.

Given a transition function B_i, and $\sigma \in \Sigma$, we identify the function $B_i(\cdot, \sigma) : [w] \to [w]$ 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 $\mathsf{A}(B_i) \triangleq \mathbb{E}_{\sigma\in\Sigma}[B_i(\sigma)]$. 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 $\sum_{v\in V_{\mathrm{acc}}} \mathsf{A}(B)[v_0, v]$. 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.

3.3 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 [1, 7, 27]. We describe it formally.

Definition 3.3 (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 3.4. *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,*

$$\left\|\mathsf{R}(A, B, k) - B^{-1}\right\| \le \left\|B^{-1}\right\| \cdot \varepsilon^k.$$

The above lemma can be used to devise an algorithm that improves the accuracy of matrix powers [1, 7, 27], as we state below. For completeness, we provide the short proof in [8, Appendix B].

Lemma 3.5. *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 of substochastic matrices $(B_{i,j})_{1\le i<j\le n}$ satisfying:*

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

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

- *R runs in $O\Big(\log^2 k + \log k \cdot \log(nT)\Big)$ space, where*

$$T = \max\left\{|A_i|, \left|(B)_{i,j}\right|\right\}$$

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 $A = A_1 = A_2 = \ldots = A_n$.

3.4 The Nisan Generator

Nisan, in his seminal work [24], 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 $|\Sigma| = O\left(\frac{nw|\Gamma|}{\varepsilon}\right)$,[4] and let $\mathcal{H} \subseteq \Sigma \to \Sigma$ with $|\mathcal{H}| = |\Sigma|^2$ be a two-universal family of hash functions. The seed for

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

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

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

recursively as follows.

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

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 $h = (h_1, \dots, h_{\log n})$ are good in the sense that $G(\Sigma, h)$ ε-fools B. Formally,

Theorem 3.6 ([24]). *Given $n, w \in \mathbb{N}$, an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and an alphabet Γ, let $G : \{0, 1\}^{d_{\mathsf{N}}} \times \Sigma \to \Sigma^n$ be the above Nisan generator, where $|\Sigma| = O\left(\frac{nw|\Gamma|}{\varepsilon\delta}\right)$ and $d_{\mathsf{N}} = O(\log n \cdot \log |\Sigma|)$. 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*

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

recalling that $\mathsf{A}(B) = \mathbb{E}_{z\in\Gamma^n}[B(z)]$.[5]

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

[5]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 $\mathbb{E}_{\sigma\in\Sigma}[B(G(h, \sigma))]$.

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 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" randomness, 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 [32] 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 $\sigma \in \Sigma$. One can then easily show that 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. One can then show that if every entry of M is represented using at most $s \ge t$ bits, 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 3.7 (following [24]). *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*

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

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 3.7 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 3.8. *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

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

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 $i \in [n]$,

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

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 $M_h^{(n)}$.

4 BACKGROUND: THE SZ 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*

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

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

The preceding lemma is an important ingredient in [32], 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 Theorem 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 [16, 18, 37] for additional discussion. Extending Theorem 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*

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

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

4.2 The SZ 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 $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, $t = \log\frac{2nw^2r_2}{\varepsilon\delta}$, $\ell = \frac{t}{2}$, $\varepsilon_{\mathsf{N}} = 2^{-2t}$, 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 $A \in \mathbb{R}^{w\times w}$, $r = r_1r_2$, $h \in \{0,1\}^{d_{\mathsf{N}}}$, and $\mathsf{Z} = (\mathsf{Z}_1,\ldots,\mathsf{Z}_{r_2}) \in \{0,\ldots,2^\ell - 1\}^{r_2}$, and proceeds as follows.

(1) Set $\widetilde{M}_0 = A$.
(2) For $i = 1,\ldots,r_2$,
   (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 $\widetilde{M}_i = \left\lfloor \widetilde{M}_{i-1}^{(2^{r_1})} - \mathsf{Z}_i\cdot 2^{-2t}J_w\right\rfloor_t$.
(3) Output $\widetilde{M}_{r_2}$.

Theorem 4.4 ([32]). *For any $w \times w$ stochastic matrix A, and integers r_1, r_2 such that $r_1r_2 = r = \log n$, the above algorithm satisfies the following. With probability at least $1-\delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$ and $\mathsf{Z} = (\mathsf{Z}_1,\ldots,\mathsf{Z}_{r_2}) \in \{0,\ldots,2^\ell-1\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}(A, r_1, r_2, h, \mathsf{Z})$ satisfies*

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

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

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.

Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ the desired confidence. Let $r \in \mathbb{N}$, and write $r = r_1r_2$ 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 3.8, the seed for Nisan's algorithm is of length

$$d_{\mathsf{N}} = O\left(r_1\cdot\left(t_1 + r_1 + \log\frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)\right) = O\left(r_1^2 + r_1\log\frac{r_2w}{\delta}\right).$$

For the shift and truncate we take $\ell = \log\frac{2w^2r_2}{\delta}$. 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

$$r_2\cdot\ell = O\left(r_2\cdot\log\frac{r_2w}{\delta}\right).$$

Finally, set $t_2 = \log\frac{16w^2r_2n}{\varepsilon\delta}$, and notice that $t_2 = \Omega(\log\frac{n}{\varepsilon})$. We stress that the key fact that unlike [32], 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 $(\mathsf{Z}_1,\ldots,\mathsf{Z}_{r_2}) \in \{0,\ldots,2^\ell-1\}^{r_2}$. 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
   $$\widetilde{M}_i = \left\lfloor \mathsf{R}\left(\widetilde{M}_{i-1}^{(1)},\ldots,\widetilde{M}_{i-1}^{(2^{r_1})}, \widetilde{M}_{i-1}, 3t_2\right) - \mathsf{Z}_i 2^{-2t_2}J_w\right\rfloor_{t_2}.[6]$$
(3) Output $\widetilde{M}_{r_2}$.

We first determine our algorithm's space complexity.

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

For the correctness, we show:

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 $\mathsf{Z} = (\mathsf{Z}_1,\ldots,\mathsf{Z}_{r_2}) \in \{0,\ldots,2^\ell-1\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \mathsf{Z})$ satisfies*

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

[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).

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

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

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 $r_1 = r_2 = \sqrt{\log n}$ and $\varepsilon, \delta \geq \frac{1}{n}$ 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 [32] 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 $\mathrm{SZ}_{\mathrm{Imp}}$ 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. We defer the formal description to the full version.

6 APPROXIMATING THE ITERATED PRODUCT

In this section (or more precisely, in [8, Section 6]), 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*

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

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

6.1 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 subsection 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 3.8, 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" [2]. We prove Theorem 6.2 in [8, Appendix C], where we also discuss the underlying technique.

6.2 Dealing with the Shifts

As discussed in subsection 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 $\mathrm{Z} \cdot 2^{-2t_2}$ where $\mathrm{Z} \sim \{0, 1, \ldots, 2^{t_2} - 1\}$.
2. For $i = 1, \ldots, r_2$,
   - (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 [8, Section 6.3]. In [8, Lemma 6.6] we prove that a shift ζ of each entry of $A_1, \ldots, A_n$ results in an error matrix $E(\zeta)$, where $0 \leq E(\zeta) \leq \zeta \cdot T$, inequalities are entry-wise, and the matrix T is defined by

$$T = J_w \sum_{k=1}^{n} A_{k+1} \cdots A_n. \qquad (1)$$

This implies that each entry of E has magnitude at most $n^2\zeta$. However, generalizing the truncation lemma, Theorem 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 [8, Lemma 6.7]. Luckily for us, it turns out that

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

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 [8, Section 6.3].

Furthermore, by taking the union bound over all the "true" matrices that are obtained 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.

We leave the full description of the algorithm, and its analysis, to the full version of the paper.

ACKNOWLEDGMENTS

Gil Cohen was funded by ERC starting grant 949499 and by ISF grant 1569/18. Ori Sberlo was funded by ERC starting grant 949499 and by ISF grant 952/18. Amnon Ta-Shma was funded by ISF grant 952/18.

REFERENCES

[1] AmirMahdi Ahmadinejad, Jonathan Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil Vadhan. 2020. High-precision estimation of random walks in small space. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1295–1306. https://doi.org/10.1109/FOCS46700.2020.00123

[2] Roy Armoni. 1998. On the derandomization of space-bounded computations. In *Randomization and approximation techniques in computer science*. LNCS, Vol. 1518. Springer, 47–59. https://doi.org/10.1007/3-540-49543-6_5

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

[4] Andrej Bogdanov, William M. Hoza, Gautam Prakriya, and Edward Pyne. 2022. Hitting sets for regular branching programs. In *Proceedings of the 37th Computational Complexity Conference (CCC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 3:1–3:22. https://doi.org/10.4230/lipics.ccc.2022.3

[5] Allan Borodin, Stephen Cook, and Nicholas Pippenger. 1983. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control* 58, 1-3 (1983), 113–136. https://doi.org/10.1016/S0019-9958(83)80060-6

[6] Mark Braverman, Anup Rao, Ran Raz, and Amir Yehudayoff. 2014. Pseudorandom Generators for Regular Branching Programs. *SIAM J. Comput.* 43, 3 (2014), 973–986. https://doi.org/10.1137/120875673

[7] Gil Cohen, Dean Doron, Oren Renard, Ori Sberlo, and Amnon Ta-Shma. 2021. Error reduction for weighted PRGs against read once branching programs. In *Proceedings of the 36th Computational Complexity Conference (CCC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 22:1–22:17. https://doi.org/10.4230/LIPIcs.CCC.2021.22

[8] Gil Cohen, Dean Doron, and Ori Sberlo. 2022. Approximating Iterated Multiplication of Stochastic Matrices in Small Space. In *Electronic Colloquium on Computational Complexity (ECCC)*.

[9] Anindya De. 2011. Pseudorandomness for permutation and regular branching programs. In *Proceedings of the 26th Computational Complexity Conference (CCC)*. IEEE, 221–231. https://doi.org/10.1109/CCC.2011.23

[10] Dean Doron, Raghu Meka, Omer Reingold, Avishay Tal, and Salil Vadhan. 2021. Pseudorandom Generators for Read-Once Monotone Branching Programs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 58:1–58:21. https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.58

[11] Michael A. Forbes and Zander Kelley. 2018. Pseudorandom Generators for Read-Once Branching Programs, in any Order. In *Proceedings of 59th Annual Symposium on the Foundations of Computer Science (FOCS)*. IEEE, 946–955. https://doi.org/10.1109/FOCS.2018.00093

[12] John Gill. 1977. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.* 6, 4 (1977), 675–695. https://doi.org/10.1137/0206049

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

[14] William Hoza. 2022. Recent Progress on Derandomizing Space-Bounded Computation. In *Electronic Colloquium on Computational Complexity (ECCC)*.

[15] William M. Hoza. 2021. Better Pseudodistributions and Derandomization for Space-Bounded Computation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 28:1–28:23. https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.28

[16] William M. Hoza and Adam R. Klivans. 2018. Preserving Randomness for Adaptive Algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 43:1–43:19. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2018.43

[17] William M. Hoza, Edward Pyne, and Salil Vadhan. 2021. Pseudorandom generators for unbounded-width permutation branching programs. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 7:1–7:20. https://doi.org/10.4230/LIPIcs.ITCS.2021.7

[18] William M. Hoza and Chris Umans. 2021. Targeted pseudorandom generators, simulation advice generators, and derandomizing logspace. *SIAM J. Comput.* (2021), STOC17–281. https://doi.org/10.1137/17M1145707

[19] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. 1994. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual Symposium on Theory of Computing (STOC)*. ACM, 356–364. https://doi.org/10.1145/195058.195190

[20] H. Jung. 1981. Relationships between Probabilistic and Deterministic Tape Complexity. In *Mathematical Foundations of Computer Science (MFCS) (LNCS, Vol. 118)*. Springer, 339–346. https://doi.org/10.1007/3-540-10856-4_101

[21] Adam R. Klivans and Dieter van Melkebeek. 2002. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.* 31, 5 (2002), 1501–1526. https://doi.org/10.1137/S0097539700389652

[22] Michal Koucký, Prajakta Nimbhorkar, and Pavel Pudlák. 2011. Pseudorandom generators for group products. In *Proceedings of the 43rd Annual Symposium on*

Theory of Computing (STOC). ACM, 263–272. https://doi.org/10.1145/1993636.1993672

[23] Raghu Meka, Omer Reingold, and Avishay Tal. 2019. Pseudorandom generators for width-3 branching programs. In *Proceedings of the 51st Annual Symposium on Theory of Computing (STOC)*. ACM, 626–637. https://doi.org/10.1145/3313276.3316319

[24] Noam Nisan. 1992. Pseudorandom generators for space-bounded computation. *Combinatorica* 12, 4 (1992), 449–461. https://doi.org/10.1007/BF01305237

[25] Noam Nisan. 1994. **RL** ⊆ **SC**. *computational complexity* 4, 1 (1994), 1–11. https://doi.org/10.1007/BF01205052

[26] Noam Nisan and David Zuckerman. 1996. Randomness is linear in space. *J. Comput. System Sci.* 52, 1 (1996), 43–52. https://doi.org/10.1006/jcss.1996.0004

[27] Edward Pyne and Salil Vadhan. 2021. Pseudodistributions that beat all pseudorandom generators. In *Proceedings of the 36th Computational Complexity Conference (CCC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 33:1–33:15. https://doi.org/10.4230/LIPIcs.CCC.2021.33

[28] Edward Pyne and Salil Vadhan. 2022. Deterministic Approximation of Random Walks via Queries in Graphs of Unbounded Size. In *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 57–67. https://doi.org/10.1137/1.9781611977066.5

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

[30] Omer Reingold, Thomas Steinke, and Salil Vadhan. 2013. Pseudorandomness for regular branching programs via Fourier analysis. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. LNCS, Vol. 8096. Springer, 655–670. https://doi.org/10.1007/978-3-642-40328-6_45

[31] Michael Saks. 1996. Randomization and derandomization in space-bounded computation. In *Proceedings of the 11th Computational Complexity Conference (CCC)*. IEEE, 128–149. https://doi.org/10.1109/CCC.1996.507676

[32] Michael E. Saks and Shiyu Zhou. 1999. $\mathbf{BP_H SPACE}(S) \subseteq \mathbf{DSPACE}(S^{2/3})$. *J. Comput. System Sci.* 58, 2 (1999), 376–403.

[33] Walter J. Savitch. 1970. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. System Sci.* 4, 2 (1970), 177–192. https://doi.org/10.1016/S0022-0000(70)80006-X

[34] Janos Simon. 1981. On tape-bounded probabilistic Turing machine acceptors. *Theoretical Computer Science* 16, 1 (1981), 75–91. https://doi.org/10.1016/0304-3975(81)90032-3

[35] Thomas Steinke. 2012. Pseudorandomness for Permutation Branching Programs Without the Group Theory. In *Electronic Colloquium on Computational Complexity (ECCC)*.

[36] Thomas Steinke, Salil Vadhan, and Andrew Wan. 2017. Pseudorandomness and Fourier-growth bounds for width-3 branching programs. *Theory of Computing* 13, 1 (2017), 1–50. https://doi.org/10.4086/toc.2017.v013a012

[37] Amnon Ta-Shma. 2013. Inverting well conditioned matrices in quantum logspace. In *Proceedings of the 45th Annual Symposium on Theory of Computing (STOC)*. ACM, 881–890. https://doi.org/10.1145/2488608.2488720