# Approximating Large Powers of Stochastic Matrices in Small Space

Gil Cohen[*]
Dept. of Computer Science
Tel Aviv University
gil@tauex.tau.ac.il

Dean Doron
Dept. of Computer Science
Ben Gurion University
deand@bgu.ac.il

Ori Sberlo[†]
Dept. of Computer Science
Tel Aviv University
orisberlo@mail.tau.ac.il

**Abstract**

We give a deterministic space-efficient algorithm for approximating powers of stochastic matrices. On input a $w \times w$ stochastic matrix $A$, our algorithm approximates $A^n$ in space $\widetilde{O}(\log n + \sqrt{\log n} \cdot \log w)$ to within high accuracy. This improves upon the seminal work by Saks and Zhou [SZ99], that requires $O(\log^{3/2} n + \sqrt{\log n} \cdot \log w)$ space, in the regime $n \gg w$.

[*]Funded by ERC starting grant 949499 and by ISF grant 1569/18.
[†]Funded by ERC starting grant 949499 and by ISF grant 952/18.

## 1 Introduction

One of the great challenges in complexity theory 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? It is widely believed that $\mathbf{BPL} = \mathbf{L}$, as indeed follows from plausible circuit lower bounds [KvM02], and in contrast to the time-bounded case, there are no known barriers for the unconditional derandomization of $\mathbf{BPL}$. Over the past decades, significant progress has been made, notably in the construction of pseudorandom generators (PRGs) for various classes of branching programs, and in space-efficient derandomization of fundamental problems in graph theory and linear algebra.

Savitch's result [Sav70] can be extended to show that any two-sided error randomized algorithm that uses $S$ space can be simulated deterministically using $O(S^2)$ space, giving $\mathbf{BPL} \subseteq \mathbf{DSPACE}(\log^2 n)$. Nisan [Nis92, Nis94] devised a time-efficient derandomization with a quadratic overhead in space, namely, $\mathbf{BPL} \subseteq \mathbf{DTISP}(\mathrm{poly}(n), \log^2 n)$. Focusing solely on space, Saks and Zhou [SZ99] cleverly built on Nisan's work to deterministically simulate two-sided error space $S$ randomized algorithms in space $O(S^{3/2})$. The state-of-the-art is a recent improvement by Hoza [Hoz21], giving a deterministic simulation in space $O(S^{3/2}/\sqrt{\log S})$, utilizing recent advances in weighted PRGs for branching programs in the low-error regime [BCG18, CL20, CDR+21, PV21].

A more general setting is where we consider derandomization of space-$S$ algorithms that use $R$ random coins, where $R$ is not necessarily assumed to be $2^{\Theta(S)}$. Nisan and Zuckerman [NZ96] proved that any space-$S$ randomized algorithm that uses $R = \mathrm{polylog}(S)$ coins can be simulated deterministically in space $O(S)$. See also [Arm98] for a different range of parameters. The distinction between $R$ and $S$ is also widely, and successfully, studied in the context of PRGs for branching program, where $w = 2^{\Theta(S)}$ corresponds to the width of the program, and $R$ corresponds to its length (for the formal definitions, see Section 2.3). For a few examples, often for more restricted classes of space-bounded computation, see [RR99, BRRY14, MRT19, DMR+21] and references therein.

**Approximating powers of stochastic matrices.** It is easy to see that approximating powers of stochastic matrices is in $\mathbf{BPL}$, by estimating the probability of a random walk over the corresponding Markov chain. Conversely, it is possible to convert any randomized space-bounded machine into a stochastic operator $A$ such that the probability the machine moves from a state $s$ to a state $t$ in $k$ steps is $A^k[s,t]$. Therefore, derandomizing space-bounded algorithms amounts to deterministically, and space-efficiently, approximating powers of $A$. Namely, for two-sided error space-$\log w$ algorithms that use $n$ bits of randomness, the task would be to approximate the entries of $A^n$ for any $w \times w$ stochastic operator $A$.

Following Savitch [Sav70], using repeated squaring we can compute $A^n$ exactly in space $O(\log n \cdot \log w)$ (ignoring bit representation issues, see Claim 2.6). When $w \ll n$, one can get better space complexity by using an algorithm based on the Cayley–Hamilton theorem.

**Theorem 1.1.** *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 1.1 uses standard linear algebra and known results from parallel circuit complexity, we are not aware of any reference in which it explicitly appears. Thus, for completeness, we give the formal details in Appendix A.

For stochastic matrices, introducing $\varepsilon > 0$ approximation error, the Saks–Zhou algorithm runs in space

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

The dependence on $\varepsilon$ was recently improved by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford, and Vadhan [AKM+20] using the Richardson iteration (see Section 2.4), obtaining

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

space.

## 1.1 Main Result

The main result of this work is an algorithm that improves upon the classical Saks-Zhou algorithm (as well as [AKM+20]) in the regime $n \gg w$.

**Theorem 1.2** (main result; see also Theorem 6.1)**.** *For any $w, n \in \mathbb{N}$, and $\varepsilon > 0$, there exists a deterministic algorithm that given a $w \times w$ stochastic matrix $A$, approximates $A^n$ 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 $w$. More precisely, our algorithm requires*

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

*space.*

Our algorithm outperforms previous results (including Theorem 1.1) whenever

$$2^{\sqrt{\log n}} \ll w \ll n.$$

For concreteness, let us take $w = 2^{\log^{\alpha} n}$ for some constant $\alpha \in (\frac{1}{2}, 1)$. Our algorithm runs in $\widetilde{O}(\log^{1/2+\alpha} n)$ space, whereas the Saks–Zhou algorithm requires $O(\log^{3/2} n)$ space, and the algorithm that is given by Theorem 1.1 requires $O(\log^{2\alpha} n)$ space.

There are two natural ways to further interpret our result.

**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 1.2 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 about the behavior of random walks.

**Space-bounded derandomization.** In the lens of derandomization, Theorem 1.2 proves that any randomized algorithm that uses $R$ random bits and $S$ space can be simulated deterministically in $O(\log R + \sqrt{\log R} \cdot S)$ space. In the regime $R = 2^{S^{1/\alpha}}$ for $\alpha < 1$, 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 $R$ random coins *in expectation*. With this modification, our simulation result holds. We make two additional remarks: (1) For $R = 2^{O(S)}$, the above modification agrees with the standard model, and (2) Taking $R \gg 2^S$ may decide languages outside $\mathbf{BPL}$, e.g., if $R = 2^{2^{O(S)}}$ then we can decide the directed connectivity problem which is not known to be in $\mathbf{BPL}$.

## 1.2 Our Algorithm

Our result is based on the beautiful Saks–Zhou algorithm which consists of two ingredients: (1) The celebrated Nisan generator, which is used as a randomized matrix exponentiation algorithm, and (2) the *shift and truncate* technique (see Section 4). By the latter, we mean subtracting a small quantity from intermediate calculations (i.e., shift), and keeping only some of the most significant digits (i.e., truncate). To approximate $A^n$ given some stochastic matrix $A$, the [SZ99] algorithm follows by using (1) to approximate the $2^{\sqrt{\log n}}$-th power, iteratively for $\sqrt{\log n}$ times, and applying (2) between consecutive iterations. Applying (2) (in tandem with a canonicalization step we discuss in Section 3) allows [SZ99] to *reuse* the randomness needed for repeated applications of Nisan generator.

We start by noting that each shift must be taken to be smaller than $\frac{1}{n}$ as the quality of approximation deteriorates as we take higher powers. A key observation of our work is that, in contrast to the *magnitude* of each shift, the amount of *randomness* the [SZ99] algorithm needs for the shifts can be taken to be independent of $n$. Concretely, choosing a random shift from the set $\left\{1 \cdot n^{-1}, \ldots, 2^{\ell} \cdot n^{-1}\right\}$ requires $\ell$ random bits, even though the shifts themselves are of magnitude $\frac{1}{n}$. Indeed, our algorithm invests roughly $\log w$ random bits for choosing the shifts and each shift is of magnitude roughly $\frac{1}{n}$.

While these parameters suffice for the analysis to work, this observation does not readily yield any improvement: Nisan generator is costly if the input matrix is represented using too many bits of precision. If we keep $\log n$ bits of accuracy for each entry, Nisan generator runs in space $\log n$, eventually resulting in an overall space complexity of $\log^{3/2} n$.[1] We would want each iteration to run in space $\sqrt{\log n} + \log w \ll \log n$, and it is unclear how to utilize the aforementioned observation in order to get an improvement upon [SZ99].

We thus employ the following approximation scheme. First, we purposely *decrease* the precision of the input matrix to the Nisan generator by truncating its entries to a precision of $\frac{1}{w}$. The output of the generator then gives us a "mild" approximation to the $2^{\sqrt{\log n}}$-th power. Then, to restore the (required) high precision approximation of $\frac{1}{n}$, we invoke the *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 (see Section 2.4). In our work, we use it to obtain a high precision approximation of matrix powers from mild approximations. It is instructive to note that although we decrease the precision before using Nisan generator, this precision is not lost since we still "keep" the untruncated matrix. In turn, the Richardson iteration combines the untruncated matrix with mild approximation of its $2^{\sqrt{\log n}}$-th power, in order to get a high-precision approximation of that power.

We provide a rough outline of our algorithm (see also Figure 1). The precise description is given in Section 5. The 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 $\widetilde{M}_0 = A$.
2. For $i = 1, \dots, \sqrt{\log n}$,
   (a) Truncate $\widetilde{M}_{i-1}$ to a precision of $\frac{1}{w}$ and denote this by $\lfloor \widetilde{M}_{i-1} \rfloor$.
   (b) Set the Nisan generator to work with accuracy $\frac{1}{w}$ and use it to approximate $\lfloor \widetilde{M}_{i-1} \rfloor^{2^{\sqrt{\log n}}}$. Note that since $\widetilde{M}_{i-1} \approx \lfloor \widetilde{M}_{i-1} \rfloor$, we get $\widetilde{M}_{i-1}^{2^{\sqrt{\log n}}} \approx \lfloor \widetilde{M}_{i-1} \rfloor^{2^{\sqrt{\log n}}}$.
   (c) Use the mild approximation obtained above to compute a high precision approximation $\mathsf{R}_i \approx \widetilde{M}_{i-1}^{2^{\sqrt{\log n}}}$ by applying the Richardson iteration. We stress that the Richardson iteration improves our approximation with respect to the previous high precision approximation $\widetilde{M}_{i-1}$ and not its truncation.
   (d) Shift $\mathsf{R}_i$ by a random shift of magnitude $\frac{1}{n}$, and truncate it to a precision of $\frac{1}{n}$. We set $\widetilde{M}_i$ to be the result of that shift and truncation.
3. Output $\widetilde{M}_{\sqrt{\log n}}$.

[1]In fact, the "inner" seed length will also be $\log^{3/2} n$, which is a bigger issue, since the space complexity can be improved by employing the sampler trick together with the INW generator instead of Nisan's.

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

Figure 1: Our Improved SZ Algorithm. "S & T" refers to "shift and truncate".

The above outline, and Figure 1, illustrate the alternating nature of the algorithm, zigzagging between a mild approximation of $\frac{1}{w}$ and a high precision approximation of $\frac{1}{n}$. 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 $\frac{1}{w}$. It is thus tempting to try and apply an additional layer of the Richardson iteration in order to improve the accuracy to an arbitrary $\varepsilon > 0$.[2] However, to apply the Richardson iteration, the initial accuracy needs to be at least $\frac{1}{n} \ll \frac{1}{w}$. 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 Preliminaries

### 2.1 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 $\ell_\infty$ norm, i.e., $\|A\|_\infty = \max_{i \in [w]} \sum_{j \in [w]} |A[i,j]|$. Clearly,

**Claim 2.1.** *For any matrix $M \in \mathbb{R}^{w \times w}$ we have that $\|M\|_\infty \le w \|M\|_{\max}$.*

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 *sub-stochastic* if its entries are nonnegative and every row sums to at most $1$, i.e., $\|A\|_\infty \le 1$.

The following claim follows by a simple induction and the triangle inequality.

[2]In fact, this was already done in [AKM+20] for the standard Saks–Zhou algorithm.

**Claim 2.2.** *Let $\|\cdot\|$ be a sub-multiplicative 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 $\left\|A^k - B^k\right\| \leq k \cdot \|A - B\|$.*

## 2.2 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 2.3** (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 $s_1, s_2 \colon \mathbb{N} \to \mathbb{N}$, 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 2.4.** *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*

$$O\left(\sum_{i=0}^{k-1} s\left(\ell_i(n)\right)\right)$$

*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 2.5** (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 2.6.** *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 2.4. (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. □

## 2.3 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 $\Sigma$, 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 $B_i$, 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 $B$. In particular, $\sum_{v \in V_{\mathsf{acc}}} \mathsf{A}(B)[v_0, v]$ denotes the probability that $B$ accepts a random input. 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 $\varepsilon$-approximate the aforementioned acceptance probability for any $v_0$ and $V_{\mathsf{acc}}$.

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.

## 2.4 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 2.7** (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 2.8.** *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,*

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

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

**Lemma 2.9.** *There exists an algorithm $\mathsf{R}$ that gets as input a sub-stochastic matrix $A \in \mathbb{R}^{w\times w}$, sub-stochastic matrices $A_1, \ldots, A_n \in \mathbb{R}^{w\times w}$, and $k \in \mathbb{N}$, and satisfies the following.*

- *If for all $i \in [n]$ we have that $\|A^i - A_i\|_\infty \le \frac{1}{4(n+1)}$, then*

$$\|\mathsf{R}(A_1, \ldots, A_n, A, k) - A^n\|_\infty \le (n+1)\cdot 2^{-k}.$$

- *$\mathsf{R}$ runs in*

$$O\left(\log^2 k + \log k \cdot \log(nT)\right)$$

  *space, where $T = \max\{|A|, |A_1|, \ldots, |A_n|\}$ is the maximum bit-complexity of the given matrices.*

# 3 Revisiting the Nisan Generator

Nisan, in his seminal work [Nis92], constructed a family of pseudorandom generators that $\varepsilon$-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" $\Sigma$, 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 $O(\log |\Sigma|) \ll d$. It turns out to be incredibly useful to divide the seed into an "offline" part $h \in \{0,1\}^{d_{\mathsf{N}}}$ which we can fix and is good with high probability, and an "online" one $\sigma \in \Sigma$ which we average over. We can summarize the properties of the Nisan generator below, where we explicitly distinguish between *accuracy* and *confidence* (as was also done in [Nis94] and in more recent works).

**Theorem 3.1** ([Nis92]). *Given $n, w \in \mathbb{N}$, an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and an alphabet $\Gamma$, 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 $d_{\mathsf{N}} = O(\log n \cdot \log |\Sigma|)$. 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 $\mathsf{A}(B) = \mathbb{E}_{z \in \Gamma^n}\left[B(z)\right]$.*[4]

In particular, the above says that if $B$ is a $[w, \Sigma]$ BP with a transition matrix $A$, we can use Nisan generator to approximate powers of $A$ by using the same layer.

**Canonicalization of BPs.** An important step in [SZ99] is to transform a given stochastic matrix (or a sub-stochastic one) into a BP, in a canonical way. We first make this notion explicit.

Given a $w \times w$ sub-stochastic 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.

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

[4] We note that by "collecting instructions", we 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 $\mathbb{E}_{\sigma \in \Sigma}\left[B(G(h,\sigma))\right]$.

**Claim 3.2.** *For a sub-stochastic 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 sub-stochastic matrix with a large bit-complexity into a BP over a small alphabet. Given a sub-stochastic $M$ 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 3.3.** *Let $M$ be a $w \times w$ sub-stochastic 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 $\mathsf{C}_t$ 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. We can now summarize the parameters of the generator as a randomized algorithm for approximating powers of matrices.

**Theorem 3.4** (following [Nis92])**.** *There exists an algorithm $\mathsf{N}$ 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 note that one can improve the overall dependence on $\delta$ using Armoni's "sampler trick" [Arm98] (see also [CL20, Appendix A]), but we will not need it.

We will often want to feed Nisan's algorithm with stochastic (or even sub-stochastic) matrices, rather than BPs. The following theorem extends upon Theorem 3.4 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 $\mathsf{N}$ for both the algorithm that gets a BP as input and for the one that gets a matrix as input.

**Theorem 3.5.** *There exists an algorithm* $\mathsf{N}$ *that gets as input:*

1. *A $w \times w$ sub-stochastic 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 $O\left(\log s + \log \frac{nw}{\varepsilon\delta}\right)$ 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 $i \in [n]$,*

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

*When we omit the parameter $t$, we implicitly set $t = s$, and then the error guarantee is simply $\varepsilon$. Also, when we set $\mathsf{N}$ 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 3.4, 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 3.3, $\|\mathsf{A}(B) - A\|_\infty \le w \cdot 2^{-t}$, and thus, due to Claim 2.2,

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

The space requirements and the bound for $d_{\mathsf{N}}$ readily follows from Claim 3.3 and Theorem 3.4. Note that when $t = s$, the canonicalization is lossless. $\square$

## 4 The Saks–Zhou Algorithm

We revisit Saks and Zhou's argument in a different terminology, which would allow us to lay the groundwork for our improved algorithm given in the next section. Toward this end, we recall the machinery of *shift and truncate*.

## 4.1 Shift and Truncate

**Definition 4.1** (truncation). *For $z \in [0,1]$ 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 sub-stochastic 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*

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

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

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

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

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

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

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 $\zeta$ satisfying Equation (1). $\square$

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 [TS13, 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*

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

*where $\zeta$ 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 $r$. (This can be assumed without any significant loss in parameters.) 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 the 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 $\zeta = (\zeta_1, \dots, \zeta_{r_2}) \in \left\{0, \dots, 2^\ell - 1\right\}^{r_2}$, 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 $\widetilde{M}_i = \left\lfloor \widetilde{M}_{i-1}^{(2^{r_1})} - \zeta_i 2^{-2t} J_w \right\rfloor_t$.
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 $\zeta = (\zeta_1, \dots, \zeta_{r_2}) \in \left\{0, \dots, 2^\ell - 1\right\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}(A, r_1, r_2, h, \zeta)$ satisfies*

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

*Moreover, $\mathsf{SZ}(A, r_1, r_2, h, \zeta)$ 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 $i \in [r_2]$,

$$M_i(\zeta) = \lfloor M_{i-1}(\zeta)^{2^{r_1}} - \zeta_i 2^{-2t} J_w \rfloor_t. \tag{2}$$

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

Next, we argue that with high probability (over $h$ and the $\zeta$-s), $\widetilde{M}_i = M_i$. Toward this end, define for each fixing of $\zeta_1, \dots, \zeta_i$,

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

It is important to note that whenever $\zeta_1, \dots, \zeta_i$ 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 $h$. By Theorem 3.5, we get that for any $i \in [r_2]$ and $\zeta_1, \dots, \zeta_i$,

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

**Claim 4.5.** *It holds that*

$$\Pr_{h,\zeta}\left[\exists j \in [r_2],\ M_j \ne \widetilde{M}_j\right] \le r_2 w^2 2^{-\ell} \le \delta.$$

*Proof.* We prove by induction on $i$ that

$$\Pr_{h,\zeta}\left[\exists j \le i,\ M_j \ne \widetilde{M}_j\right] \le \left(\delta_{\mathsf{N}} + w^2 2^{-\ell}\right) \cdot i.$$

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

$$E = \left\{ (h, \zeta) : \exists j < i \text{ such that } M_j \ne \widetilde{M}_j \text{ or } h \notin \mathrm{GOOD}_{i-1,\zeta} \right\}.^{5}$$

Next, we write

$$\begin{aligned}
\Pr_{h,\zeta}\left[\exists j \le i, M_j \ne \widetilde{M}_j\right] &= \Pr[E]\Pr\left[\exists j \le i, M_j \ne \widetilde{M}_j \mid E\right] + \Pr[\neg E]\Pr\left[M_i \ne \widetilde{M}_i \mid \neg E\right] \\
&\le \Pr[E] + \Pr\left[M_i \ne \widetilde{M}_i \mid \neg E\right] \\
&\le \Pr\left[\exists j < i, M_j \ne \widetilde{M}_j\right] + \Pr\left[h \notin \mathrm{GOOD}_{i-1,\zeta}\right] + \Pr\left[M_i \ne \widetilde{M}_i \mid \neg E\right].
\end{aligned}$$

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 $\delta_{\mathsf{N}}$, so it suffices to show that

$$\Pr_{h,\zeta}\left[M_i \ne \widetilde{M}_i \mid \forall j < i \quad M_j = \widetilde{M}_j \quad \text{and} \quad h \in \mathrm{GOOD}_{i-1,\zeta}\right] \le w^2 2^{-\ell}.$$

In fact, we shall show that for any *fixed* $\zeta_1, \dots, \zeta_{i-1}$ and $h$ satisfying the above conditioning, we have

$$\Pr_{\zeta_i}\left[M_i \ne \widetilde{M}_i\right] \le w^2 2^{-\ell}.$$

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

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

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

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

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

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

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

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. □

Next, we handle the accuracy guarantee.

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

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

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

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

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

and so by Claim 2.1, $\left\|M_i - M_{i-1}^{2^{r_1}}\right\|_\infty \le 2^{-t+1} w$. Write

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

By the induction's hypothesis and Claim 2.2, 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

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

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

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

For the space complexity, by Theorem 3.5, $\mathsf{N}$ takes $O\left(r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)$ space and the rest of the operations per iteration are absorbed within the latter term. This yields space complexity of $r_2 \cdot O\left(r_1 + \log \frac{w}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right) = r_2 \cdot O\left(\log \frac{nw}{\varepsilon\delta}\right)$. $\square$

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 $\ell_\infty$ 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 Our Improved Algorithm

In this section, and in the next one, 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 1.2. Toward this end, we devise a randomized algorithm which is derandomized in Section 6.

### 5.1 The Randomized Algorithm

We now formally describe our 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. Set $\varepsilon_{\mathsf{N}} = 2^{-2r_1}$, $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, $\ell = \log \frac{2w^2 r_2}{\delta}$, $t_1 = 4r_1 + \log w$, $t_2 = \log \frac{16 w^2 r_2 n}{\varepsilon\delta}$, and

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

Without the loss of generality we may assume that the input matrix $A$ is also given to us using $t_2$ digits of precision. Additionally,

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

The algorithm $\mathsf{SZ}_{\mathsf{Imp}}$ gets as input $A \in \mathbb{R}^{w \times w}$ [6], $r = r_1 r_2$, $h \in \{0,1\}^{d_{\mathsf{N}}}$, and $\zeta = (\zeta_1, \ldots, \zeta_{r_2}) \in \{0, \ldots, 2^\ell - 1\}^{r_2}$, and proceeds as follows.

[6] We assume that each entry of $A$ is represented using $t_2$ bits of accuracy.

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

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

   (a) Compute $\left(\widetilde{M}_{i-1}^{(1)}, \widetilde{M}_{i-1}^{(2)}, \dots, \widetilde{M}_{i-1}^{(2^{r_1})}\right) = \mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}\left(\widetilde{M}_{i-1}, h, 2^{r_1}, t_1\right)$.

   (b) Compute $\widetilde{M}_i = \left\lfloor \mathsf{R}\left(\widetilde{M}_{i-1}^{(1)}, \dots, \widetilde{M}_{i-1}^{(2^{r_1})}, \widetilde{M}_{i-1}, 3t_2\right) - \zeta_i 2^{-2t_2} J_w \right\rfloor_{t_2}$.[7]

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

Before delving into the analysis, let us briefly discuss our parameters.

- The truncation parameters $t_2 > t_1$: In Item 2b we truncate all but the first $t_2$ bits of our approximation. The parameter $t_2$ determines the accuracy of the algorithm, and is governed by $n$. The second truncation occurs implicitly in Item 2a where in the canonicalization step we truncate all but the first $t_1$ bits before applying the Nisan generator. The parameter $t_1$, which is governed by $w$ and $r_1$, does not affect the accuracy due to the Richardson iteration performed right after.

- The parameters of the Nisan generator $\varepsilon_{\mathsf{N}}$ and $\delta_{\mathsf{N}}$ correspondingly denote the accuracy and confidence parameters we apply $\mathsf{N}$ with. We abbreviate $\mathsf{N}_{\varepsilon_{\mathsf{N}},\delta_{\mathsf{N}}}(A, h, 2^{r_1}, t_1)$ with $\mathsf{N}(A, h)$.

- The "shift" parameter $\ell$: The amount of randomness we invest in the shifts, which also determines the probability in which the shifts are successful. Note that we set $\ell \ll t_2$, and in particular $\ell$ does not depend on $n$ and the accuracy parameter $\varepsilon$. In contrast, in [SZ99], $\ell = \Omega(t)$ (see Section 4).

We first determine our algorithm's space complexity.

**Lemma 5.1.** *Computing* $\mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \zeta)$ *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 itself the composition of three functions:

[7]The Richardson iteration may output a matrix which is not sub-stochastic. 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 $(h, \zeta)$.

- The Nisan generator $\mathsf{N}$: By Theorem 3.5, this takes

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

  space.

- Richardson Iteration: By Lemma 2.9, 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 2.4 we can sum the above and multiply by $r_2$, obtaining our desired overall space complexity. $\square$

Before continuing with correctness, we need to make sure our parameters satisfy the following constraints.

- To apply Corollary 4.3, we need

$$\ell < t_2. \tag{5}$$

- The truncation parameters satisfy

$$t_1 < t_2. \tag{6}$$

- In order to apply the Richardson iteration (Lemma 2.9) we need to satisfy

$$\varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1} \le \frac{1}{4(2^{r_1}+1)},\ 2^{-t_2} \le \frac{1}{2^{r_1}+1}. \tag{7}$$

## 5.2 Proof of Correctness

Throughout the analysis we shall assume that Equations (5) to (7) hold.

**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 $\zeta = (\zeta_1, \ldots, \zeta_{r_2}) \in \{0, \ldots, 2^\ell - 1\}^{r_2}$, the output $\widetilde{M}_{r_2} = \mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \zeta)$ satisfies*

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

*where $r = r_1 r_2$ and $n = 2^r$. Moreover, $\mathsf{SZ}_{\mathsf{Imp}}(A, r_1, r_2, h, \zeta)$ 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 rounding, similar to the analysis in Section 4. That is, $M_0 = A$, and for each $i \in [r_2]$,

$$M_i(\zeta) = \left\lfloor M_{i-1}(\zeta)^{2^{r_1}} - \zeta_i 2^{-2t_2} J_w \right\rfloor_{t_2}. \tag{8}$$

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

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

$$\mathrm{GOOD}_{i,\zeta} = \left\{ h \in \{0,1\}^{d_{\mathsf{N}}} : \forall j \le 2^{r_1}, \left\| M_i^j - M_i^{(j)} \right\|_\infty \le \varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1} \right\}, \tag{9}$$

where $\left(M_i^{(1)}, \ldots, M_i^{(2^{r_1})}\right) = \mathsf{N}(M_i, h)$. (Note that here we feed $\mathsf{N}$ with $M_i$ and not $\widetilde{M}_i$, similar to what we did in Section 4.) By Theorem 3.5, we get that for any $i \in [r_2]$ and $\zeta = (\zeta_1, \ldots, \zeta_i)$,

$$\Pr_{h \in \{0,1\}^{d_{\mathsf{N}}}} \left[ h \in \mathrm{GOOD}_{i,\zeta} \right] \ge 1 - \delta_{\mathsf{N}}. \tag{10}$$

**Claim 5.3.** *It holds that*

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

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

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

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

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

and again we have

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

By the induction's hypothesis the first term is at most $(\delta_{\mathsf{N}} + w^2 2^{-\ell}) \cdot (i-1)$, and by Equation (10) the second term is at most $\delta_{\mathsf{N}}$. Thus, it suffices to show that

$$\Pr_{h,\zeta} \left[ M_i \ne \widetilde{M}_i \mid \forall k < i, M_k = \widetilde{M}_k \ \text{ and } \ h \in \mathrm{GOOD}_{i-1,\zeta} \right] \le w^2 2^{-\ell}.$$

We show that for any fixed $\zeta_1, \dots, \zeta_{i-1}$ and $h$ satisfying the conditioning, we have

$$\Pr_{\zeta_i}\left[M_i \neq \widetilde{M}_i\right] \le w^2 2^{-\ell}.$$

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

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

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

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

Using Lemma 2.9 and the guarantee of Equation (7), we get

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

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

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

which simply means that $M_i = \widetilde{M}_i$, recalling the definition of $M_i$ from Equation (8). This completes the inductive step. □

For the accuracy guarantee, we have

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

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

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

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 $\left\|\widetilde{M}_{r_2} - A^n\right\| \le \varepsilon$. The space requirement was established in Lemma 5.1. □

We note that by averaging over all seeds (namely, $h$ and the $\zeta$-s), taking $\delta = \varepsilon \approx \frac{1}{w}$, we would get a deterministic approximation, and this was also done in [SZ99]. However, we can get a much better accuracy, and this is the content of the next section.

## 6 A High-Accuracy Deterministic Approximation

In this section we prove our main result, Theorem 6.1, giving a high-accuracy deterministic algorithm for approximating $A^n$. Note that $\mathsf{SZ}_{\mathsf{Imp}}$ does not provide us with one good approximation but requires a seed. In [SZ99], Saks and Zhou averaged over the outputs of all seeds. When $\varepsilon \approx \delta$, this would give an $O(\varepsilon)$-approximation deterministic algorithm.

Recall that Richardson iteration forces us to take $\varepsilon \approx \frac{1}{n}$. The dependence of Theorem 5.2 on $\varepsilon$ is only double-logarithmic, and so taking a tiny $\varepsilon$ does not deteriorate the space complexity by much. The dependence on $\delta$, however, is logarithmic. Thus, to gain any improvement we cannot afford to take $\delta \approx \varepsilon$ as would be the case if we "mixed" the $\delta$ fraction of bad matrices together with the accurate ones. Our key idea here is as follows: Instead of averaging, we iterate the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm over all $(h,\zeta)$-s and take the entry-wise *median* of the outputs. This approach only requires us to take $\delta = O(1)$. Toward that end, given a set of matrices $\{A_1,\dots,A_m\}$, 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 $i \in [m]$.

Formally, the $\mathsf{SZ}^{+}_{\mathsf{Imp}}$ algorithm 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, we get that 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 $d_{\mathsf{N}}$ 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 $\zeta_1,\dots,\zeta_{r_2}$ is given by

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

Then,

1. For $i = 1,\dots,\log n$, compute

$$\widetilde{M}_{2^i} = \operatorname*{median}_{h,\zeta}\ \mathsf{SZ}_{\mathsf{Imp}}\left(\lfloor A\rfloor_{t_2}, \sqrt{i}, \sqrt{i}, h, \zeta\right),$$

   where we take the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm with accuracy $\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}$ and confidence $\delta_{\mathsf{SZ}_{\mathsf{Imp}}}$.[8]

[8] 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.

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

**Theorem 6.1.** *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
$$O\left( (\log n + r_2 \log w) \log\log \frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2 \log \frac{1}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2 \left( \log\log \frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}} \right)^2 \right)$$
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
$$d_{\mathsf{N}} + |\zeta| + O\left(\log t_2 w^2\right) = O\left(\log n + \sqrt{\log n} \cdot \log w\right).$$

- 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 $\mathsf{R}$ 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 2.9, 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 2.3).

We now proceed with the correctness. By Theorem 5.2, for at least $\frac{3}{4}$ of the $(h,\zeta)$-s, we have
$$\left\|\mathsf{SZ}_{\mathsf{Imp}}(\lfloor A\rfloor_{t_2},\sqrt{i},\sqrt{i},h,\zeta) - \lfloor A\rfloor_{t_2}^{2^i}\right\|_{\max} \le \left\|\mathsf{SZ}_{\mathsf{Imp}}(\lfloor A\rfloor_{t_2},\sqrt{i},\sqrt{i},h,\zeta) - \lfloor A\rfloor_{t_2}^{2^i}\right\|_{\infty} \le \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}},$$
and so for at least $\frac{3}{4}$ of the $(h,\zeta)$-s we get that for all $(a,b)\in[w]^2$,
$$\left|\mathsf{SZ}_{\mathsf{Imp}}\left(\lfloor A\rfloor_{t_2},\sqrt{i},\sqrt{i},h,\zeta\right)[a,b] - \lfloor A\rfloor_{t_2}^{2^i}[a,b]\right| \le \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$
Thus, for all $(a,b)\in[w]^2$,
$$\left|\left(\operatorname*{median}_{h,\zeta}\ \mathsf{SZ}_{\mathsf{Imp}}\left(\lfloor A\rfloor_{t_2},\sqrt{i},\sqrt{i},h,\zeta\right)\right)[a,b] - \lfloor A\rfloor_{t_2}^{2^i}[a,b]\right| \le \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$
This is true for all indices $(a,b)$ and all $i\in[\log n]$ and so by Claim 2.1, for all $i\in[\log n]$,
$$\left\|\widetilde{M}_{2^i} - \lfloor A\rfloor_{t_2}^{2^i}\right\|_{\infty} \le w\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$
By Claim 2.2, $\left\|A^j - \lfloor A\rfloor_{t_2}^{j}\right\|_{\infty} \le jw2^{-t_2}$, and applying Claim 2.2 again for multiplication of $\log n$ matrices, we get that for every $j\le n$,
$$\left\|\widetilde{M}_j - A^j\right\|_{\infty} \le \left\|\widetilde{M}_j - \lfloor A\rfloor_{t_2}^{j}\right\|_{\infty} + \left\|\lfloor A\rfloor_{t_2}^{j} - A^j\right\|_{\infty} \le \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} w\log n + nw2^{-t_2} \le \frac{1}{4(n+1)}.$$
Using Lemma 2.9, we obtain
$$\left\|\mathsf{R}(\widetilde{M}_1,\ldots,\widetilde{M}_n,A,k) - A^n\right\|_{\infty} \le (n+1)\cdot 2^{-k} \le \varepsilon,$$
which completes the proof. □

## References

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

[AKM$^+$20] Amir Mahdi Ahmadinejad, Jonathan Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil Vadhan. High-precision estimation of random walks in small space. In *61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1295–1306. IEEE, 2020.

[Arm98] Roy Armoni. On the derandomization of space-bounded computations. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, 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 *50th Annual Symposium on Theory of Computing (STOC 2018)*, pages 353–362. ACM, 2018.

[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.

[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 *36th Computational Complexity Conference (CCC 2021)*, pages 22:1–22:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[CL20] Eshan Chattopadhyay and Jyun-Jie Liao. Optimal error pseudodistributions for read-once branching programs. In *35th Computational Complexity Conference (CCC 2020)*, pages 25:1–25:27. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[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 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

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

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

[HK18] William M. Hoza and Adam R. Klivans. Preserving randomness for adaptive algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*. 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 2021)*. 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.

[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 *51st Annual Symposium on Theory of Computing (STOC 2019)*, 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 *36th Computational Complexity Conference (CCC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[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.

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

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

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

[TS13] Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *45th Annual Symposium on Theory of Computing (STOC 2013)*, pages 881–890. ACM, 2013.

## A Spectral Algorithm for Matrix Powering

In this section we prove Theorem 1.1. 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 one 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 to run 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 2.6. 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 Proof of Lemma 2.9

In this section, for completness, we prove Lemma 2.9.

*Proof of Lemma 2.9.* The algorithm constructs the following pair of block matrices which consists of $(n+1) \times (n+1)$ blocks of $w \times w$ matrices. For $0 \le i, j \le n$,

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

The algorithm then outputs the matrix $\mathsf{R}(A, B, k)$ as given in Definition 2.7.

The space complexity of the algorithm follows by Claim 2.6. As for the correctness, first observe that

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

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

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

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

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

The correctness now follows by Lemma 2.8.

□