Tel Aviv University
Raymond and Beverly Sackler Faculty of Exact Sciences
Blavatnik School of Computer Science

Error Correcting Codes and Space Bounded Derandomization

A thesis submitted for the degree of
Doctor of Philosophy
by
Ori Sberlo

Thesis supervisors: Prof. Gil Cohen,
Prof. Amnon Ta-Shma.

Submitted to the Senate of Tel Aviv University
September 2022

Contents

1 On Codes Decoding Constant Fraction of Random Errors 1

1.1 Introduction . . . 1
1.2 Preliminaries . . . 3
1.3 Our Results . . . 7
1.4 Appendix . . . 15

2 Introduction to Space Bounded Derandomization 19

2.1 Computational Complexity . . . 19
2.2 Space Bounded Computation . . . 19
2.3 Randomized Computation . . . 22
2.4 Branching Programs . . . 23
2.5 Pseudorandomness and Derandomization . . . 27
2.6 Brief Account of Space Bounded Derandomization . . . 29
2.7 Efficient Implementations of PRGs Against ROBPs . . . 30

3 Error Reduction For Weighted PRGs Against ROBPs 33

3.1 Background . . . 33
3.2 Our Result . . . 34
3.3 Overview . . . 35
3.4 Preliminaries . . . 39
3.5 The Richardson Iteration . . . 39
3.6 Richardson Iteration Is a WPRG . . . 40
3.7 The Construction . . . 42
3.8 Proof of Correctness . . . 44

4 Approximating Powers of Stochastic Matrices in Small Space 45

4.1 Background . . . 45
4.2 Our Result . . . 49
4.3 Overview . . . 51
4.4 Preliminaries . . . 54
4.5 Revisiting the Saks–Zhou Framework . . . 55
4.6 Algorithm Outline, Correctness, and Complexity . . . 60
4.7 Proof of Correctness . . . 62
4.8 A High-Accuracy Deterministic Approximation . . . 64
4.9 Appendix . . . 67

5 Approximating Products of Stochastic Matrices in Small Space **69**
5.1 Background 69
5.2 Our Result 70
5.3 Overview 70
5.4 Algorithm Outline, Correctness, and Complexity 74
5.5 Perturbing the Input 76
5.6 Proof of Correctness 78
5.7 The Space Complexity 82
5.8 Appendix 83

Abstract

The first part of this thesis (Chapter 1) is dedicated to *error correcting codes* in Shannon's noise model. We show that every binary linear error correcting code that can be reliably transmitted over some Binary-Memoryless-Symmetric-Channel (BMSC), can be reliably transmitted over any BMSC with sufficiently small Bhattacharyya parameter. Based on [KKM+17] we conclude that every doubly-transitive code with moderate minimum distance can can be reliably transmitted over any BMSC with sufficiently small Bhattacharyya parameter. An important corollary of our result is that the Reed-Muller code, a fundamental error correcting code, can recover from a constant fraction of random noise over every BMSC, which was a long standing open problem[1].

The second part is dedicated to space-bounded derandomization. First, we focus on pseduorandomness giving an error reduction procedure for pseudorandom generators (PRGs) against space-bounded computation. Our reduction has optimal dependence on the error parameter. Unfortuantely, the procedure does not produce a PRG, but rather a *weighted* PRG (WPRG). This result was independently obtained by Pyne, and Vadhan [PV21].

Lastly, we discuss a problem which in some sense captures the problem of space-bounded derandomization. Ignoring technicalities, approximating the T-th power of $2^S \times 2^S$ stochastic matrices is equivalent to derandomizing algorithms that run in time T, and use S space. Saks and Zhou [SZ99] gave an algorithm for this problem that runs in

$$O((S + \log T)\sqrt{\log T})$$

space. We improve upon their algorithm, and achieve space complexity of $O(S\sqrt{\log T})$. Our improvement only applies whenever $S \ll \log T$, and in particular only applies to algorithms that their running time is much larger than the space they use. Such algorithms necessarily do not halt, and so one has to consider the non-halting model in which the algorithm is allowed not to halt with vanishing probability, and the running time is defined as the expected running time instead. What about approximating the iterated product of stochastic matrices

$$A_1, A_2, \cdots, A_n \in \mathbb{R}^{w \times w}?$$

The best algorithm that was known for this problem is due to the aforementioned work of Saks and Zhou. However, in the case $w \ll n$ nothing better was known. We make progress on this natural problem devising an algorithm that runs in roughly

$$O(\sqrt{\log n}\log w)$$

space. In order to illustrate the difference, consider the case

$$w = 2^{\sqrt{\log n}},$$

namely we have n matrices of dimension $2^{\sqrt{\log n}}$. For this setting of parameters, the Saks–Zhou algorithm yields a space complexity of $O(\log^{3/2} n)$, and in fact, an $O(\log^{3/2} n)$ space algorithm can be derived easily without the Saks–Zhou algorithm. In contrast, our algorithm achieves nearly-optimal space complexity of $O(\log n \log\log n)$.

The first part is based on a joint work with Jan Hązła and Alex Samorodnitsky [HSS21], and the second part on joint works with Gil Cohen, Dean Doron, Oren Renard, and Amnon Ta-Shma [CDR+21, CDSTS22].

[1]In a later work [RP21], it was shown that the Reed-Muller code achieve capacity, namely are optimal, over every BMSC.

Chapter 1

On Codes Decoding Constant Fraction of Random Errors

1.1 Introduction

In this work we study binary linear codes over binary memoryless symmetric channels and their weight distribution.

A binary linear error correcting code is a linear subspace $V \subseteq \mathbb{F}_2^n$. The subspace V should have the property that given an erroneous version of $v \in V$ one can extract from it some information on v (v is usually referred to as a codeword). This is possible because while $v \in \mathbb{F}_2^n$ is an n-bit vector, it belongs to a subspace smaller than the entire space, or equivalently because v contains redundancies. The amount of redundancy in V is captured by the ratio $R(V) = \frac{\dim(V)}{n}$ which is a fundamental property of a code called the *rate*.

In order to make the above concrete one has to formally define the manner in which errors are induced and what information on $v \in V$ should be retrieved. One such simple model was proposed by Hamming [Ham50] in which corruptions are adversarial and we seek to recover the entire original codeword. The behaviour in this setting is completely determined by the *minimum distance* which is the Hamming distance between the two closest codewords. Other interesting models include list decoding, deletion channels, locally decodable codes and more. In this work we shall focus on Shannon's model in which corruptions are induced randomly and we seek to recover the original codeword with high probability. Perhaps the simplest types of corruption are the binary erasure channel (BEC) and the binary symmetric channel (BSC). In the BEC(p), every bit is independently erased[1] with fixed probability $p \in [0, 1]$ and in the BSC(p) every bit is flipped with probability $p \in [0, 1/2]$. These two channels belong to a larger family of *binary memoryless symmetric channels* (BMS channels). Memoryless means that the noise is independent for every coordinate and symmetric roughly means that the corruption of 1 and 0 is symmetric. For instance, had we flipped 0 with probability p_0 and flipped 1 with probability p_1 for $p_0 \neq p_1$ then this would not have been a symmetric channel. Another important example of a BMS channel is the binary additive Gaussian white noise channel BAWGN(σ) where we add a Gaussian noise $z \sim \mathcal{N}(0, \sigma^2)$ to every coordinate independently.

In his seminal work [Sha48], Shannon provided an upper bound on the amount of noise a code can tolerate over any BMS channel. He proved that a code $V \subseteq \mathbb{F}_2^n$ can be decoded successfully over a channel $\mathcal{W}$ only if its rate $R(V)$ does not exceed the *channel capacity* $C(\mathcal{W})$. For instance,

[1]By erasing a coordinate we mean to replace it with '?'. Do not confuse it with deletion in which the codeword length varies.

$C(\mathrm{BEC}(p)) = 1 - p$ and so codes with rate larger than $1 - p$ cannot recover from p fraction of random erasures. Codes that achieve this bound are called capacity achieving (with respect to a channel $\mathcal{W}$). One may wonder if such codes even exist and indeed, random codes as well as random linear codes are capacity achieving. There are also explicit constructions of capacity achieving codes, most notably *polar codes* introduced by Arıkan [Ar 09] that achieve capacity for any BMS channel with efficient encoding and decoding.

It is often the case that good error correcting codes come from algebraic structures. Examples include the Reed–Muller codes and the BCH codes which are among the oldest binary codes (we shall soon discuss the Reed–Muller codes). Yet, none of these were known to achieve capacity even for the BEC. This was resolved by Kudekar, Kumar, Mondelli, Pfister, Şaşoğlu and Urbanke [KKM+17] who proved that a binary linear code which satisfy some symmetry property called *double-transitivity* (see Definition 1.3.8) achieves capacity for the BEC. In particular, the aforementioned codes satisfy this symmetry property and hence are capacity achieving for the BEC. Unfortunately, their technique seems less amenable to other channels such as the BSC or the BAWGN. Our result extends theirs to arbitrary BMS channels, albeit not to the capacity limit, in the following way.

Theorem 1.1.1 ([[HSS21]). *(Informal)] Let $V \subseteq \mathbb{F}_2^n$ be a doubly transitive code with rate $R = R(V)$, and minimum distance $d(V) = \Omega(n^\alpha)$. Then, V decodes errors on $\mathrm{BSC}(p(R, \alpha))$, $\mathrm{BAWGN}(\sigma(R, \alpha))$ where $p(R, \alpha)$, $\sigma(R, \alpha)$ are some explicit functions depending on α, R.*

This can be extended, in an appropriate way, to arbitrary BMS channels.

For details see Section 1.3.2. This is achieved by going through the *weight distribution* of a code which is the sequence enumerating how many codewords $v \in V$ there are with a given number of ones. For linear codes, this determines how far apart the codewords are and hence serves as a good statistic to measure how much noise the code can tolerate. Indeed, sufficiently strong bounds on the weight distribution of a linear code imply that it can decode errors on a given BMS channel.

Recently, bounds on the weight distribution of codes that achieve capacity for the BEC were given in [Sam19]. By combining the two approaches of [KKM+17, Sam19] we derive slightly stronger bounds on the weight distribution of such codes which in turn implies good performance for general BMS channels.

In fact, we establish a general method that applies to any linear code that is good enough at recovering from random erasures. We give exact definitions later, but in the theorem below $P_{\mathrm{B}}(\mathcal{W}, V)$ is the probability of failure in recovering a codeword using the optimal (so-called maximum a posteriori, or block-MAP) decoder for a code V on a channel $\mathcal{W}$. The Bhattacharyya parameter $Z(\mathcal{W}) \in [0, 1]$ is a property of a channel, with a lower value of $Z(\mathcal{W})$ intuitively corresponding to a less noisy channel:

Theorem 1.1.2 ([HSS21]). *Let V be a binary linear code with dimension $\dim(V) = k$, minimum distance d and block-MAP error probability over $\mathrm{BEC}(\lambda)$ less than $1/k$. Then, for any BMS channel $\mathcal{W}$,*

$$P_{\mathrm{B}}(\mathcal{W}, V) < 2 \left(\frac{Z(\mathcal{W})}{2^\lambda - 1} \right)^d .$$

Therefore, a linear code that recovers from random erasures also decodes errors on BMS channels with small enough Bhattacharayya parameter. As discussed in Remark 1, our assumptions on the minimum distance and error probability over the BEC are mild. While below we discuss a specific application to Reed–Muller codes, we note that our result can be applied more generally, e.g., for

BCH codes, LDPC codes or polar codes for the BEC. On the other hand, we stress that we are only concerned with MAP decoding, and in general there is no reason to expect that it is efficient.

Our result applies to an important family of binary codes — the Reed–Muller codes. Reed–Muller (RM) codes were introduced by Muller [Mul54] and rediscovered shortly after by Reed [Ree54]. The RM code $\mathrm{RM}(m, r)$ is defined[2] by all the evaluation vectors of multi-linear polynomials over $\mathbb{F}_2$ with m variables and degree at most r. Due to [KKM+17], we know that constant rate RM codes[3] achieve capacity for the BEC. What about the BSC? BAWGN? It is considered plausible that RM codes achieve capacity for those channels as well [ASY20]. However, prior to this work it was unknown whether a constant rate RM code can correct even a tiny constant fraction of random errors. In the regime of non-constant rate there are some strong results. For rates approaching 0, Abbe, Shpilka and Wigderson [ASW15] proved that $\mathrm{RM}(m, r)$ achieve capacity[4] for the BSC if $r = o(m)$ which was later improved to $r = m/70$ in [SS20]. The latter also provides some results on the BSC for any $r < (1/2 - o(\sqrt{\log m/m}))m$. For rates approaching 1 the best result is again due to [ASW15] who proved that the Reed–Muller code $\mathrm{RM}(m, r)$ achieves capacity for the BSC if $r > m - O(\sqrt{m/\log m})$. For constant rates, [AHN20] shows that subcodes of constant rate RM codes where an arbitrarily small constant fraction of basis elements was deleted correct a constant fraction of random errors.

In contrast, applying Theorem 1.1.2 to the result from [KKM+17] we obtain an unconditional result for constant rate RM codes, albeit falling short of the capacity threshold:

Theorem 1.1.3 ([HSS21])**.** *For any $0 < R < 1$, a family of RM codes with rates approaching R decodes errors on any BMS channel with $Z(\mathcal{W}) < 2^{1-R} - 1$.*

In particular, it decodes errors on the channels $\mathrm{BSC}(p)$ with $p < 1/2 - \sqrt{2^{-R}(1 - 2^{-R})}$ and $\mathrm{BAWGN}(\sigma)$ with $\sigma^2 < -\frac{1}{2\ln(2^{1-R}-1)}$.

There is also a broader consequence of our result. Throughout the literature there are (somewhat varying) definitions of "asymptotically good" families of codes. In Hamming's model, a family of codes is usually considered good if it has both constant rate $R = \Omega(1)$ and linear minimum distance $d = \Omega(n)$. Therefore, being good is a property of the code. On the other hand, in Shannon's model, it is sometimes said [Mac99] that a family of codes $\{V_n\}$ is good for a given channel $\mathcal{W}$ if it has constant rate $R = \Omega(1)$ and vanishing block-MAP error probability $P_{\mathrm{B}}(\mathcal{W}, V_n) = o_n(1)$. Our result shows that for linear codes with moderate minimum distance the notion of a good code is in fact independent of the channel. On the one hand, by a degradation argument (see Lemma 1.3.7), a code which is good for a BMS channel is also good for $\mathrm{BEC}(\lambda)$ for some $\lambda > 0$. On the other hand, by our results, a family of linear codes which is good for the BEC is also good for all BMS channels up to a certain Bhattacharyya parameter (and also above a certain capacity, see Remark 2 and Corollary 1.3.5).

1.2 Preliminaries

In this section we give definitions that are most relevant to our results. Throughout we take $\log(\cdot)$ to denote the binary logarithm and $H(\cdot)$ the Shannon entropy. We also let $h_2(x) = -x \log x - (1 - x) \log(1 - x)$ for $0 \le x \le 1$ be the binary entropy function.

[2]It is possible to generalize RM codes to arbitrary fields. This is sometimes referred to as generalized RM codes [DGMW70].

[3]Constant rate RM codes means that $\lim_{m\to\infty} R(\mathrm{RM}(m, r)) \in (0, 1)$ where $r = r(m)$ (e.g, $r(m) = m/2$).

[4]One should be careful about what "capacity-achieving" means in the regimes of rates approaching 0 or 1. See [ASW15] for more details.

1.2.1 Error Correcting Codes

Basic Definitions

Let $V \subseteq \mathbb{F}_2^n$ be a binary linear code of block length n. Define the following:

- Rate: $R(V) = \frac{\dim(V)}{n}$.
- Weight: For $v \in \mathbb{F}_2^n$ we define $\mathrm{wt}(v) = |\{j \mid v_j = 1\}|$.
- Minimum Distance: $d(V) = \min_{v \in V \setminus \{0^n\}} \mathrm{wt}(v)$.
- Weight Distribution: The sequence $(|\{v \in V \mid \mathrm{wt}(v) = i\}|)_{i=0}^n$ is the weight distribution of V.
- Dual: Let $V^\perp = \{u \in \mathbb{F}_2^n \mid \forall v \in V, u^T v = 0\}$ be the dual code of V which is a linear subspace of dimension $n - \dim(V)$.
- Restriction: Given $S \subseteq [n]$ define $V_S \subseteq \mathbb{F}_2^S$ as the restriction of V to the coordinates in S.

In the asymptotic setting, we consider families of codes $\{V_n\}$ where $n \in N \subseteq \mathbb{N}$ comes from an infinite set of block lengths. We define the rate of the family as $R = \lim_{n\to\infty} R(V_n)$ (if it exists). When $R = 0$ we say that the family $\{V_n\}$ has vanishing rate, and non-vanishing rate otherwise. Also, when $R \in (0,1)$ we say that the family $\{V_n\}$ has constant rate.

Reed–Muller Codes

Definition 1.2.1. *The Reed–Muller code* $\mathrm{RM}(m,r) \subseteq \mathbb{F}_2^n$ *is a linear code with block length* $n = 2^m$ *consisting of all evaluations of multilinear polynomials over* $\mathbb{F}_2$ *with* m *variables and degree at most* r*. That is, for every such polynomial* f *there is a codeword* $(f(a))_{a \in \mathbb{F}_2^m}$.

It is known that $\mathrm{RM}(m,r)$ has minimum distance 2^{m-r} and rate $R(\mathrm{RM}(m,r)) = 2^{-m}\sum_{j=0}^{r}\binom{m}{j}$. Moreover, usually $r = r(m)$ is a function of m, and in this case the family $\{\mathrm{RM}(m, r(m))\}_m$ has non-vanishing rate smaller than 1 if and only if $r(m) = \frac{m}{2} \pm O(\sqrt{m})$. Therefore, a family of Reed–Muller codes with constant rate $R \in (0,1)$ (i.e, $R \neq 0,1$) has minimum distance $n^{1/2 \pm o_n(1)}$.

1.2.2 Shannon's Model

Channels

Abstractly, a binary channel is defined by two conditional probability distributions $p_{Y|X}$ where $X \in \{0,1\}$ is binary and $Y \in \mathcal{Y}$ taken from some alphabet.

Definition 1.2.2 (Binary Erasure Channel)**.** *Define the* $\mathrm{BEC}(\lambda)$ *over* $\mathcal{Y} = \{0,1,?\}$ *by* $p_{Y|X}(? \mid 0) = p_{Y|X}(? \mid 1) = \lambda$*,* $p_{Y|X}(0 \mid 0) = p_{Y|X}(1 \mid 1) = 1 - \lambda$*.*

Definition 1.2.3 (Binary Symmetric Channel)**.** *Define the* $\mathrm{BSC}(p)$ *over* $\mathcal{Y} = \{0,1\}$ *by* $p_{Y|X}(1 \mid 0) = p_{Y|X}(0 \mid 1) = p$*,* $p_{Y|X}(0 \mid 0) = p_{Y|X}(1 \mid 1) = 1 - p$*.*

Definition 1.2.4 (Binary Additive White Gaussian Noise Channel)**.** *In this case we interpret the input as* $X \in \{-1,+1\}$ *and take* $\mathrm{BAWGN}(\sigma)$ *over* $\mathcal{Y} = \mathbb{R}$ *as*

$$Y|X \sim \mathcal{N}(X, \sigma^2)\ .$$

In this work we focus on the class of binary memoryless symmetric channels:

Definition 1.2.5 (BMS Channel)**.** *A BMS channel is a binary channel for which there exists an involution π on $\mathcal{Y}$ such that the distribution $p_{Y|X=0}$ is equal to $p_{\pi(Y)|X=1}$.*

We remark that $\mathrm{BEC}(\lambda), \mathrm{BSC}(p), \mathrm{BAWGN}(\sigma)$ are all BMS channels.

When we say that a codeword $v \in \mathbb{F}_2^n$ is transmitted over a BMS channel $\mathcal{W}$, we always assume that the bits are transmitted over n independent instances of $\mathcal{W}$ (that is, in a memoryless fashion).

MAP Decoding

Let $\mathcal{W}$ be a BMS channel and $n \in \mathbb{N}$. Assume we transmit a uniformly random codeword $X \in V$ and denote $Y \in \mathcal{Y}^n$ the output of the channel. We define the *maximum a posteriori* (MAP) block decoding by

$$\hat{x}^{\mathrm{MAP}}(y) = \arg\max_{v \in V} p_{X|Y}(v \mid y) \ ,$$

breaking ties in an arbitrary way. For instance, for the BSC one can easily verify that $\hat{x}^{\mathrm{MAP}}(y)$ is simply the codeword $v \in V$ closest to y in the Hamming distance. Similarly, we can define the bit-MAP decoding

$$\hat{x_i}^{\mathrm{MAP}}(y) = \arg\max_{x_i \in \{0,1\}} p_{X_i|Y}(x_i \mid y).$$

The error of block/bit-MAP decoding is the probability that these two decoders are incorrect.

Definition 1.2.6 (Block Error)**.** *The block-MAP error probability is defined by $P_{\mathrm{B}}(\mathcal{W}, V) = \mathbb{P}[\hat{x}^{\mathrm{MAP}}(Y) \neq X]$.*

Definition 1.2.7 (Bit Error)**.** *For $i \in [n]$ define the error of bit i via $P_{\mathrm{b,i}}(\mathcal{W}, V) = \mathbb{P}[\hat{x_i}^{\mathrm{MAP}}(Y) \neq X_i]$. The bit-MAP error probability is defined by $P_{\mathrm{b}}(\mathcal{W}, V) = \frac{1}{n} \cdot \sum_{i=1}^{n} P_{\mathrm{b,i}}(\mathcal{W}, V)$.*

Definition 1.2.8 (Decoding errors)**.** *For a family of linear codes $\{V_n\}$, we say that V_n decodes errors on a BMS channel $\mathcal{W}$ if $\lim_{n\to\infty} P_{\mathrm{B}}(\mathcal{W}, V_n) = 0$.*

Decomposition of BMS Channels

We present a known characterization of BMS channels which is useful for our presentation. We stick to a concise treatment very similar to Appendix A in [ACGP21] with a more complete one, e.g., in Chapter 4 of [RU08].

Let X be uniform in $\{0,1\}$ and consider a BMS channel $\mathcal{W} : \{0,1\} \to [0,1/2] \times \{0,1\}$ that maps a bit X to a pair (P, X') satisfying two conditions: First, P is independent of X and second, conditioned on P, X' is distributed according to $\mathrm{BSC}(P)$. In other words, for every transmitted bit the decoder sees its noisy copy together with information that the bit was flipped with probability P.

It is known that any BMS channel is equivalent to a mixture of BSC channels as described above. Accordingly, a BMS channel is fully characterized by the distribution of P. For example, $\mathrm{BSC}(p)$ has deterministic $P = p$ and $\mathrm{BEC}(\lambda)$ has $P = 1/2$ with probability λ and $P = 0$ otherwise. With that characterization in mind, we define the following quantities:

Definition 1.2.9 (Channel properties)**.** *Let $\mathcal{W}$ be a BMS channel. We let its:*

- Capacity *to be $C(\mathcal{W}) = 1 - \mathbb{E}\, h_2(P)$.*
- Bhattacharyya parameter *to be $Z(\mathcal{W}) = 2\,\mathbb{E}\sqrt{P(1-P)}$.*

- (Single bit) error probability *to be* $P_e(\mathcal{W}) = \mathbb{E}\, P$.

Note that $P_e(\mathcal{W})$ is the probability of error of the MAP decoder given a transmission of one uniform bit over $\mathcal{W}$ (this is because such decoder decodes a pair (P, X) to X, which was flipped with probability $P \leq 1/2$). We also remark that our definition of capacity for BMS channels is equivalent to the standard definition from information theory.

EXIT Functions

Extrinsic Information Transfer (EXIT) functions were originally introduced by ten Brink [TB99]. Based on his work, it was later observed that these EXIT functions are intimately related to the question of achieving capacity. Indeed, they played a key role in the proof that doubly transitive codes (see Definition 1.3.8) achieve capacity over the BEC under bit-MAP decoding [KKM+17].

Definition 1.2.10 (EXIT Function)**.** *Let $V \subseteq \mathbb{F}_2^n$ be a binary linear code. Define the EXIT function of V by*

$$h(\lambda) = \frac{1}{n} \cdot \sum_{i=1}^{n} H(X_i | (Y_1, \ldots, Y_{i-1}, Y_{i+1}, \ldots, Y_n)),$$

where $X = (X_1, \ldots, X_n)$ is a uniformly random codeword in V and $Y = (Y_1, \ldots, Y_n)$ is the result of transmitting X over $\mathrm{BEC}(\lambda)$.

We now list several key properties of the EXIT function.

Lemma 1.2.11. *Let $V \subseteq \mathbb{F}_2^n$ be a binary code. Then,*

- *Monotonicity: h is increasing and $h(0) = 0$, $h(1) = 1$.*
- *Area Theorem: $\int_0^1 h(\lambda)\mathrm{d}\lambda = R(V)$.*
- *Duality: Denote by $h^{\perp}(\lambda)$ the EXIT function of $V^{\perp}$ then*
$$h^{\perp}(\lambda) = 1 - h(1 - \lambda).$$
- *$n \cdot \int_0^{\lambda} h(\lambda) = H(X|Y)$ where X is a random uniform codeword in V and Y is the result of transmitting X over the* $\mathrm{BEC}(\lambda)$ *channel.*
- *For every $\lambda \in (0, 1)$*
$$\begin{aligned} n \cdot \int_0^{\lambda} h(\lambda) &= \lambda \cdot n - \mathop{\mathbb{E}}_{S \sim \lambda}[\dim(V_S^{\perp})] \\ &= \dim(V) - \mathop{\mathbb{E}}_{S \sim \lambda}[\dim(V_{S^c})], \end{aligned}$$
where $S \sim \lambda$ means that $i \in S$ with probability λ independently for every $i \in [n]$.

For proofs and further information please see chapter 3.14 in [RU08].

1.2.3 Boolean Analysis

We introduce basic notions from boolean analysis that we use in our proofs. Let $f : \{0, 1\}^n \to \mathbb{R}$ be a function from the boolean hypercube to the reals. We will use the Walsh–Fourier decomposition $f(x) = \sum_S \widehat{f}(S)\chi_S(x)$, where $\chi_S(x) = \prod_{i \in S}(-1)^{x_i}$ and $\widehat{f}(S) = \mathbb{E}_x f(x)\chi_S(x)$.

Definition 1.2.12. $\|f\|_2 = \sqrt{\mathbb{E}_x f(x)^2}$.

Lemma 1.2.13 (Parseval's Identity)**.** *For any $f, g : \{0, 1\}^n \to \mathbb{R}$ we have $\mathbb{E}_x f(x)g(x) = \sum_S \widehat{f}(S)\widehat{g}(S)$. In particular, $\|f\|_2 = \sum_S \hat{f}(S)^2$.*

Noise Operator

For $x \in \{0,1\}^n$ and $-1 \le \rho \le 1$, let $y \sim N_\rho(x)$ be a random element of $\{0,1\}^n$ with each coordinate y_i being i.i.d equal to x_i with probability $(1+\rho)/2$ and flipped with probability $(1-\rho)/2$.

Definition 1.2.14 (Noise Operator)**.** *Let $f : \{0,1\}^n \to \mathbb{R}$ and $\rho \in [-1,1]$. Define the function $T_\rho f : \{0,1\}^n \to \mathbb{R}$ by*

$$T_\rho f(x) = \mathop{\mathbb{E}}_{y \sim N_\rho(x)} f(y).$$

Lemma 1.2.15. $\widehat{T_\rho f}(S) = \rho^{|S|} \cdot \hat{f}(S)$.

An Inequality on Noisy Functions

The following inequality is the main technical tool on which our results are based [Sam20]. In order to state it, we need the notion of the conditional expectation of a function.

Definition 1.2.16. *Let $f : \{0,1\}^n \to \mathbb{R}$ and $S \subseteq [n]$ be a subset of coordinates. We define another function $\mathbb{E}(f|S)(x) = \mathbb{E}_{y:y_S = x_S} f(y)$.*

We shall only state the theorem for the ℓ_2 norm as that is all we are going to use (for the general statement see Theorem 1.1 in [Sam20]).

Theorem 1.2.17 ([Sam20])**.** *Let $f : \{0,1\}^n \to \mathbb{R}^{\geqslant 0}$ be a non-negative function, and $\rho \in (0,1)$. Then,*

$$\log \|T_\rho f\|_2 \leqslant \mathop{\mathbb{E}}_{S \sim \lambda(\rho)} \log \|\mathbb{E}(f \mid S)\|_2,$$

where $\lambda(\rho) = \log(1+\rho^2)$ and $S \sim \lambda$ is a random subset S of $[n]$ in which each element is included independently with probability λ.

Theorem 1.2.17 can be compared to the classical hypercontractive inequality $\|T_\rho f\|_2 \le \|f\|_{1+\rho^2}$, see also [Sam19] for a more extensive discussion.

1.3 Our Results

Our main contribution is in realizing that one can combine [KKM$^+$17] with [Sam20] as well as other techniques from coding theory to obtain a rather general understanding about the performance of a given binary linear code on various BMS channels.

1.3.1 Coding on BMS Channels

Our main technical result relates the weight distribution of a linear code to its decoding properties on the BEC:

Theorem 1.3.1 ([HSS21])**.** *Let V be a linear code, $0 \le \lambda \le 1$ and $(a_0, \ldots, a_n)$ be the weight distribution of V. Then,*

$$\log \sum_{i=0}^{n} a_i \cdot (2^\lambda - 1)^i \le H(X|Y) \, , \tag{1.1}$$

Here X and Y are random variables such that X is a random uniform codeword in V and Y is the result of transmitting X over the channel BEC(λ).

Theorem 1.3.1 is a reformulation of Proposition 1.3 in [Sam19] using a well-known identity. The proof of Theorem 1.3.1 appears in Section 1.4.1.

Corollary 1.3.2. *Let V be a binary linear code with dimension k and weight distribution $(a_0, \ldots, a_n)$ then*

$$\sum_{i=0}^{n} a_i (2^\lambda - 1)^i \le 2^{k \cdot P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V)} ,$$
$$\sum_{i=0}^{n} a_i (2^\lambda - 1)^i \le 2^{n \cdot P_{\mathrm{b}}(\mathrm{BEC}(\lambda), V)} .$$

In particular, if $k \cdot P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V) < 1$ or $n \cdot P_{\mathrm{b}}(\mathrm{BEC}(\lambda), V) < 1$, then $a_i < 2(2^\lambda - 1)^{-i}$ for every i.

Proof. Recall that for a linear code on the BEC if the codeword can be uniquely recovered depends only on the erasure pattern. Therefore, if the block-MAP decoder fails on $Y = y$, then $H(X|Y = y) > 0$. Since $H(X|Y = y) \le k = \dim V$ in any event, the right-hand side of (1.1) can be bounded by

$$H(X|Y) \le k \cdot P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V) . \tag{1.2}$$

On the other hand, by the chain rule there also holds a bound in terms of the bit-error probability

$$\begin{aligned} H(X|Y) &\le \sum_{i=1}^{n} H(X_i|Y) \quad (1.3) \\ &= \sum_{i=1}^{n} P_{\mathrm{b,i}}(\mathrm{BEC}(\lambda), V) = n \cdot P_{\mathrm{b}}(\mathrm{BEC}(\lambda), V). \end{aligned}$$
□

The proof of Theorem 1.1.2 is based on the following well-known bound (see, e.g., Lemma 4.67 in [RU08] and also [GWW07] for the tighter version):

Theorem 1.3.3 (Bhattacharyya Bound)**.** *Let V be a linear code and $(a_0, a_1 \ldots)$ be its weight distribution. Then for any BMS channel $\mathcal{W}$,*

$$P_{\mathrm{B}}(\mathcal{W}, V) \leqslant \sum_{i=1}^{n} a_i \cdot Z(\mathcal{W})^i .$$

In order to make our argument self-contained, we provide a proof for Theorem 1.3.3 in Section 1.4.2.

Proof of Theorem 1.1.2. We use the Bhattacharyya bound, the minimum distance and Corollary 1.3.2 to conclude that

$$\begin{aligned} P_{\mathrm{B}}(\mathcal{W}, V) &\le \sum_{i=1}^{n} a_i Z(\mathcal{W})^i = \sum_{i=d}^{n} a_i Z(\mathcal{W})^i \\ &\le \left(\frac{Z(\mathcal{W})}{2^\lambda - 1}\right)^d \sum_{i=d}^{n} a_i (2^\lambda - 1)^i < 2 \left(\frac{Z(\mathcal{W})}{2^\lambda - 1}\right)^d . \end{aligned}$$
□

Recall that we say that a family of codes $\{V_n\}$ decodes errors on channel $\mathcal{W}$ if its block-MAP error probability is vanishing, i.e., $\lim_{n \to \infty} P_{\mathrm{B}}(\mathcal{W}, V_n) = 0$.

Applying Theorem 1.1.2 to the BSC(p) and BAWGN(σ) yields the following:

Corollary 1.3.4. *Let $\{V_n\}$ be a family of linear codes with dimensions $\dim(V_n) = k_n \to \infty$ that satisfies $P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V_n) < 1/k_n$ for large n. Then:*

1. *$\{V_n\}$ decodes errors on any BMS channel $\mathcal{W}$ with $Z(\mathcal{W}) < 2^\lambda - 1$.*
2. *$\{V_n\}$ decodes errors on $\mathrm{BSC}(p)$ as long as*
   $p < \frac{1}{2} - \sqrt{2^{\lambda-1}(1-2^{\lambda-1})}$.
3. *$\{V_n\}$ decodes errors on $\mathrm{BAWGN}(\sigma)$ as long as*
   $\sigma^2 < -\frac{1}{2\ln(2^\lambda-1)}$.

Proof. The first point is immediate from Theorem 1.1.2, with the additional observation that $k_n \to \infty$ together with

$$P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V_n) < 1/k_n$$

imply that the minimum distance d_n grows to infinity with n and therefore indeed $P_B < 2(Z(\mathcal{W})/(2^\lambda - 1))^{d_n}$ vanishes. The other points now follow substituting known formulas $Z(\mathrm{BSC}(p)) = 2\sqrt{p(1-p)}$ and $Z(\mathrm{BAWGN}(\sigma)) = \exp(-1/2\sigma^2)$ (see, e.g., Examples 4.128–4.130 in [RU08]). □

A graphical illustration of the functions from Corollary 1.3.4 is provided in Figure 1.1. Theorem 1.1.3 follows by Corollary 1.3.4 and the fact that constant rate RM codes achieve capacity on the BEC [KKM$^+$17].

Remark 1. *We note that the requirement $P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V_n) < 1/k_n$ is not much stronger than $P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V_n) = o(1)$. In particular, by Theorem 5.2 in [TZ00], if the minimum distance of a linear code satisfies $d_n = \omega(\log n)$, then $P_{\mathrm{B}}(\mathrm{BEC}(\lambda), V_n) = o(1)$ implies $P_{\mathrm{B}}(\mathrm{BEC}(\lambda'), V_n) = o(n^{-c})$ for every $\lambda' < \lambda$ and $c > 0$.*

Remark 2. *Since among BMS channels with fixed capacity $C(\mathcal{W})$ the Bhattacharyya coefficient is maximized by the BSC (see Problem 4.60 in [RU08]), a family of linear codes that decodes errors on $\mathrm{BEC}(\lambda)$ also decodes errors on all BMS channels with capacity $C(\mathcal{W}) > C(\mathrm{BSC}(p)) = 1-h(p)$ where $p = p(\lambda)$ is given in Corollary 1.3.4. Similarly, a BMS channel $\mathcal{W}$ can be degraded to $\mathrm{BSC}(P_e(\mathcal{W}))$, where $P_e(\mathcal{W})$ is the (one-bit) error probability of $\mathcal{W}$. Therefore, a family of linear codes that decodes errors on $\mathrm{BEC}(\lambda)$ also decodes errors on all BMS channels with $P_e(\mathcal{W}) < P_e(\mathrm{BSC}(p)) = p$ with $p = p(\lambda)$ as above.*

Finally, we have that a family of linear codes that decodes errors well enough on any BMS channel $\mathcal{W}$ also decodes errors on a range of other BMS channels:

Corollary 1.3.5. *Let $\{V_n\}$ be a family of linear codes with dimensions $k_n \to \infty$ satisfying $P_{\mathrm{B}}(\mathcal{W}, V_n) < 1/k_n$ for large n on some BMS channel $\mathcal{W}$. Then, $\{V_n\}$ decodes errors on any BMS channel $\mathcal{W}'$ satisfying*

$$Z(\mathcal{W}') < 4^{P_e(\mathcal{W})} - 1\ .$$

To prove Corollary 1.3.5, we need the notion of channel degradation:

Definition 1.3.6. *Let $\mathcal{W} : \{0,1\} \to \mathcal{Y}$ and $\mathcal{W}' : \{0,1\} \to \mathcal{Y}'$ be two BMS channels. We say that $\mathcal{W}$ can be degraded to $\mathcal{W}'$ if there exists a channel $\mathcal{V} : \mathcal{Y} \to \mathcal{Y}'$ such that $\mathcal{W}'$ is the composition of $\mathcal{W}$ and $\mathcal{V}$.*

We now use the following folklore fact:

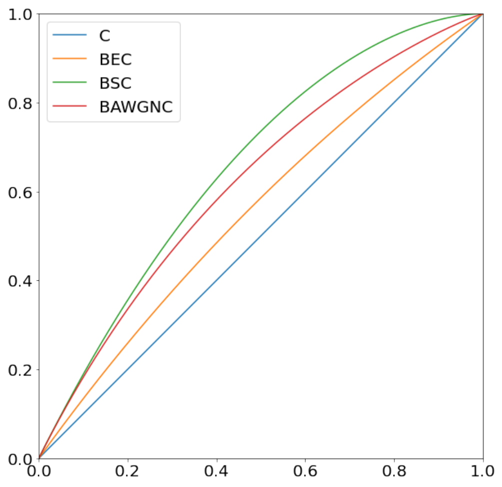


Figure 1.1: An illustration of the results in Corollary 1.3.4. Assume we are given a family of linear codes that decodes errors on the channel $\mathrm{BEC}(\lambda)$ with capacity $C(\mathrm{BEC}(\lambda)) = 1 - \lambda$. Then, by Corollary 1.3.4 it is also good for the BSC and the BAWGN exceeding certain capacities. These capacities are plotted here as functions of $C(\mathrm{BEC}(\lambda))$.
For reference, we also plot the identity function. The difference between identity and the BSC and the BAWGN curves represents the "loss of capacity" resulting when Corollary 1.3.4 is applied.
As another reference point, the graph labeled "BEC" shows this loss of capacity if our argument is applied to the BEC channel.

Lemma 1.3.7. *If $\mathcal{W}$ is a BMS channel, then* $\mathrm{BEC}(2 \cdot P_e(\mathcal{W}))$ *can be degraded to $\mathcal{W}$.*

The proof of Lemma 1.3.7 can be found, e.g., as Lemma 4.80 in [RU08]. In short, it follows from the decomposition of BMS channels into BSC that we described in Section 1.2, an easily checked fact that $\mathrm{BEC}(2p)$ can be degraded to $\mathrm{BSC}(p)$, and the fact that a convex combination of BEC channels is itself a BEC channel.

Proof of Corollary 1.3.5. Let $\{V_n\}$ be the family of linear codes from the statement and $\mathcal{W}$ a BMS channel. By Lemma 1.3.7, $\mathrm{BEC}(2 \cdot P_e(\mathcal{W}))$ can be degraded to $\mathcal{W}$. Clearly, that gives

$$P_{\mathrm{B}}(\mathrm{BEC}(2 \cdot P_e(\mathcal{W})), V_n) \leq P_{\mathrm{B}}(\mathcal{W}, V_n) < 1/k_n$$

(since a decoder for V_n on $\mathrm{BEC}(2 \cdot P_e(\mathcal{W}))$ can use the channel $\mathcal{V}$ from the definition of degradation to simulate $\mathcal{W}$ and invoke the MAP decoder for $\mathcal{W}$). The proof is concluded by an invocation of Corollary 1.3.4. □

1.3.2 Doubly Transitive Codes

In [KKM+17] it was proved that any *doubly transitive* binary linear code (see definition below) achieves capacity for the BEC by proving that its corresponding EXIT function has a *sharp threshold* (i.e., rises quickly from nearly 0 to nearly 1). Recall that $h(\lambda)$ is a monotone function increasing from 0 to 1 (see Lemma 1.2.11). Thus, if $h(\lambda)$ has a sharp threshold then this transition has to occur around $\lambda = 1 - R(C)$ by area considerations. In fact, $h(\lambda)$ has a sharp threshold if and only if V achieves capacity for the BEC. In order to prove that $h(\lambda)$ has a sharp threshold they use tools from boolean analysis which we shall soon cover.

Definition 1.3.8. *The permutation group of a binary linear code $V \subseteq \mathbb{F}_2^n$ is the group of all permutations that the code is invariant under. Namely,*

$$G = \{\pi \in S_n \mid \forall v \in V,\ \pi(v) \in V\}.$$

We say that V is doubly transitive if G is, i.e for all i, j, k distinct there exists $\pi \in G$ such that $\pi(i) = i,\ \pi(j) = k$.

Throughout this section, $V \subseteq \mathbb{F}_2^n$ is a doubly transitive code and $h(p)$ denotes its EXIT function. The main technical tool in proving that $h(p)$ has a sharp threshold is the following general theorem on monotone sets which are *sufficiently symmetric.*

Definition 1.3.9. *Let $\Omega \subseteq \{0,1\}^n$ then:*

- *We say Ω is monotone if $x \in \Omega$ and $x \leqslant y$ (i.e, $\forall i,\ x_i \leqslant y_i$) implies $y \in \Omega$.*
- *We denote $\mu_p(\Omega) = \sum_{x\in\Omega} p^{|\{i:x_i=1\}|}(1-p)^{|\{i:x_i=0\}|}$.*

Theorem 1.3.10 ([FK96] (Informal))**.** *Let $\Omega \subseteq \{0,1\}^n$ be a monotone set which is sufficiently symmetric. Then*

$$\frac{\mathrm{d}\mu_p(\Omega)}{\mathrm{d}p} \geqslant (c(p) - o_n(1)) \cdot \ln(n) \cdot \mu_p(\Omega)(1 - \mu_p(\Omega)),$$

where $c(p) = \frac{1-2p}{p(1-p)\ln\left(\frac{1-p}{p}\right)}$[5]*.*

Remark 3. *In [FK96] the above was proved with a different constant. The bound above with $c(p)$ was obtained in [Ros05].*

In the terminology of boolean analysis, the EXIT function of doubly transitive code $h(p)$ is the μ_p-measure of some monotone set $\Omega \subseteq \{0,1\}^n$, and the *sufficiently symmetric* requirement in the above theorem is fulfilled since V is doubly transitive. Thus, by applying Theorem 1.3.10 we obtain

$$h'(p) \geqslant c(p) \cdot \ln(n) \cdot h(p)(1 - h(p)). \tag{1.4}$$

The following implications of Theorem 1.3.10 appeared in [KKM+17].

Lemma 1.3.11. *Let $h(\lambda)$ be the EXIT function of a doubly transitive linear code $V \subseteq \mathbb{F}_2^n$, and $p_c = h^{-1}(1/2)$ then*

$$h(p_c - \epsilon) \leqslant n^{-k(p_c)\cdot\epsilon}, \tag{1.5}$$

where $k(p) = \begin{cases} c(p) - o_n(1) & p < 1/2 \\ c(1/2) - o_n(1) & p \geqslant 1/2 \end{cases}$.

Proof. The constant $c(p)$ in Equation (1.4) is decreasing on the interval $(0, 1/2)$ and attains its global minimum on $[0,1]$ at $p = 1/2$. Thus, by Equation (1.4) we get

$$\forall p \leqslant h^{-1}(1/2),\ h'(p) \geqslant k(p)\ln(n) \cdot h(p)(1 - h(p)).$$

It is known that for any monotone function $h : [0,1] \to [0,1]$ increasing from 0 to 1 the above inequality implies that $h(h^{-1}(1/2) - \epsilon) \leqslant \exp(-k(p)\ln(n)\epsilon)$ (for more details see Lemma 34 in [KKM+17]). □

[5] At $p = 1/2$ the function $c(p)$ has a removable discontinuity and so we define $c(1/2) = \lim_{p\to 1/2} c(p) = 2$.

As a consequence of Lemma 1.3.11 and of the second bullet in Lemma 1.2.11 one gets the following (Proposition 11 in [KKM$^+$17]).

Lemma 1.3.12. *Let $h_n(\lambda)$ be the EXIT function of a family of doubly transitive linear codes $\{V_n\}$ of rate R then $\lim_{n\to\infty} h_n^{-1}(1/2) = 1 - R$.*

For more details see section III in [KKM$^+$17].

Now we present another approach for bounding $h(p)$.

Lemma 1.3.13. *Let $h(\lambda)$ be the EXIT function of a doubly transitive linear code $V \subseteq \mathbb{F}_2^n$. Then for every $p \in [0,1]$ and $t \geqslant 1$ we have $h(\lambda^t) \leqslant h(\lambda)^t$.*

Proof. For a doubly transitive code we have $h(p) = \mu_p(\Omega)$ for some monotone set $\Omega \subseteq \{0,1\}^n$. By Lemma 2.7 in [EKL19], for a monotone set Ω and $t \geqslant 1$ it holds that $\mu_{p^t}(\Omega) \leqslant \mu_p(\Omega)^t$. □

Theorem 1.3.14 ([[HSS21]). *] Let $\{V_n\}$ be a family of doubly transitive binary linear codes of rate $R \in (0,1)$, and $\mathcal{W}$ be a BMS channel. Denote by $\alpha = \liminf_{n\to\infty} \frac{\log d(V_n)}{\log n}$. Then, V can decode errors on $\mathcal{W}$ if $Z(\mathcal{W}) < 2^{\lambda(R,\alpha)} - 1$ where*

$$\lambda(R,\alpha) = \begin{cases} (1-R) - \frac{1-\alpha}{k(1-R)} & \text{if } \psi(R,\alpha) \geqslant 0 \\ \left(\frac{(1-R)/e}{-W_{-1}(-(1-R)/e)}\right)^{\frac{1-\alpha}{k(1-R)\cdot(1-R)}} & \text{otherwise.} \end{cases},$$

where $\psi(R,\alpha) = (1-R) + \frac{1-R}{W_{-1}(-(1-R)/e)} - \frac{1-\alpha}{k(1-R)}$ and $W_{-1}(z)$ is the inverse of the function $y = xe^x$ at the interval $x \in (-1/e, 0)$ for $y \leqslant -1$ (a.k.a the negative branch of the Lambert function).

Remark 4. *It is not clear why the two cases of $\lambda(R,\alpha)$ coincide at $\psi(R,\alpha) = 0$, but one can verify this easily using simple identities of the Lambert function.*

For instance, plugging $\alpha = 0.5$ and $R = 0.8$ into Theorem 1.3.14 we get $\lambda(R,\alpha) \approx 0.0331$. This implies that any family $\{V_n\}$ of doubly transitive codes with rate $R = 0.8$ and minimum distance $d(V_n) = \Omega(\sqrt{n})$ can decode errors from BEC(0.023) under block-MAP decoding.

Proof. Let us denote $(a_0, a_1, \ldots, a_n)$ the weight distribution of V_n, $d = d(V_n)$, and $\lambda = \lambda(R,\alpha)$. By Theorem 1.3.3 we have

$$\begin{aligned} P_{\mathrm{B}}(\mathcal{W}, V_n) &\leqslant \sum_{i=d}^{n} a_i \cdot Z(\mathcal{W})^i \\ &= \sum_{i=d}^{n} a_i \cdot \left(\frac{Z(\mathcal{W})}{2^\lambda - 1}\right)^i \cdot (2^\lambda - 1)^i \\ &\leqslant \left(\frac{Z(\mathcal{W})}{2^\lambda - 1}\right)^d \cdot \sum_{i=d}^{n} a_i \cdot (2^\lambda - 1)^i. \end{aligned}$$

The last inequality follows because by assumption $\frac{Z(\mathcal{W})}{2^\lambda - 1} < 1$. Using Theorem 1.3.1 we have

$$\sum_{i=d}^{n} a_i \cdot (2^\lambda - 1)^i \leqslant 2^{H(X|Y)},$$

where X and Y are random variables such that X is a random uniform codeword in V and Y is the result of transmitting X over the channel $\mathrm{BEC}(\lambda)$. Then by Lemma 1.2.11 we get

$$P_{\mathrm{B}}(\mathcal{W}, V_n) \leqslant \left(\frac{Z(\mathcal{W})}{2^{\lambda}-1}\right)^{d} \cdot 2^{n \cdot h(\lambda)}. \tag{1.6}$$

Note that $\frac{Z(\mathcal{W})}{2^{\lambda}-1}$ is some positive constant smaller than 1 by assumption. We conclude that it suffices to show that $n \cdot h(\lambda) = o(d(V_n))$. Moreover, in our notation $d(V_n) = \Omega(n^{\alpha \pm o_n(1)})$, and so a bound of $n \cdot h(\lambda) = O(n^{\beta})$ with constant $\beta < \alpha$ suffices. This means that we want λ so that $h(\lambda) < n^{-\beta}$ with $\beta > 1-\alpha$. It remains to verify that our choice of $\lambda = \lambda(R, \alpha)$ satisfies this.

We have two ways in which we can bound $h(\lambda)$: The additive bound of Lemma 1.3.11, and the multiplicative bound of Lemma 1.3.13. It is easy to see that the additive bound is stronger for values close to the critical value $p_c = h^{-1}(1/2)$, and on the other hand the multiplicative bound is better for values that are further away from p_c. Moreover, the additive bound is limited to $\epsilon < p_c$. Applying first the additive bound with parameter ϵ and then the multiplicative bound with parameter $t \geqslant 1$ we get

$$h((p_c-\epsilon)^t) \leqslant h(p_c-\epsilon)^t \leqslant n^{-t\epsilon k(p_c)}.$$

We are going to choose λ of the form $\lambda = (p_c-\epsilon)^t$ for some valid choice of ϵ, t so that $t \cdot \epsilon \cdot k(p_c) > 1-\alpha$. Under this constraint, we want λ to be as large as possible. We are now going to cheat slightly. Instead of solving the optimization problem for λ under the constraint $t \cdot \epsilon \cdot k(p_c) > 1-\alpha$ we will solve it for $t \cdot \epsilon \cdot k(p_c) = 1-\alpha$. This suffices to prove the theorem, e.g by repeating the argument with λ' such that $Z(\mathcal{W}) < \lambda' < \lambda(R, \alpha)$ and applying Lemma 1.3.13. The optimal value of λ under this constraint will be shown to be $\lambda(R, \alpha) - o_n(1)$.

The above yields the following optimization problem.

$$\max_{t \geqslant 1, \epsilon \leqslant p_c} \{(p_c-\epsilon)^t\}, \quad tk(p_c)\epsilon = (1-\alpha). \tag{1.7}$$

The solution to this optimization problem is given by

$$p_{opt} = \begin{cases} p_c - \frac{1-\alpha}{k(p_c)} & \frac{1-\alpha}{k(p_c)} \leqslant p_c + \frac{p_c}{W_{-1}(-p_c/e)} \\ \left(\frac{p_c/e}{-W_{-1}(-p_c/e)}\right)^{\frac{1-\alpha}{k(p_c)\cdot p_c}} & otherwise. \end{cases}$$

See Section 1.4.4 for proof. Recall that by Lemma 1.3.12 we have $p_c = 1 - R - o_n(1)$ and hence $p_{opt} = \lambda(R, \alpha) - o_n(1)$ as claimed. □

Remark 5. *It is was shown in [KKM+17] that any family of binary linear doubly transitive codes $\{V_n\}$ such that $\liminf_{n\to\infty} \frac{\log d(V)}{\log n} = 1$ achieves capacity for the BEC under block-MAP decoding (Theorem 21 in [KKM+17]). Combining this with Theorem 1.1.2 implies Theorem 1.3.14 for the special case $\alpha = 1$.*

Discussion on Reed–Muller Codes

It is interesting to see that Theorem 1.3.14 is inferior to Theorem 1.1.3 in the case of Reed–Muller codes. In fact, recall that Reed–Muller codes of constant rate have minimum distance of roughly $n^{1/2}$, are doubly transitive, and yet plugging $\alpha = 1/2$ in Theorem 1.3.14 does not yield the parameters of Theorem 1.1.3. Rather, the parameters of Theorem 1.1.3 correspond to the case $\alpha = 1$ in Theorem 1.3.14. This means that RM codes perform better on BMS channels than might be expected from their minimal distance as expected from Theorem 1.3.14. Let us try to explain this phenomenon. We give two explanations. First, in [KKM+16] Kudekar, Kumar, Mondelli, Pfister, and Urbanke proved that for any constants $z, \beta \in (0,1)$ it holds that

$$\sum_{i=1}^{n^{1-\beta}} a_i \cdot z^i = o_n(1),$$

where $(a_0, a_1, \ldots, a_n)$ is the weight distribution of the Reed–Muller code (Lemma 4 in [KKM+16]). Hence, we can discard the first $n^{1-o_n(1)}$ terms in Equation (1.6) and continue the argument as in Theorem 1.3.14 but with $\alpha = 1$. This means, in a well-defined sense, that the Reed–Muller codes *effectively* have distance $n^{1-o(1)}$. An alternative explanation, which in fact follows the original approach in [KKM+17], is to use stronger results on sharp thresholds since RM codes are more than just doubly transitive. Specifically, the bound of Lemma 1.3.12 can be improved to $n^{-\Omega(\log\log n)}$ in the case of the Reed–Muller codes [BK97]. Using this improved bound and following the same approach as in Theorem 1.3.14 also leads to Theorem 1.1.3.

1.3.3 Weight Distribution of Codes

Coming back to weight distributions, and using the argument from [Sam19], Theorem 1.3.1 can be applied to the dual code $V^\perp$, resulting in a different bound on the weight distribution:

Proposition 1.3.15. *Let $V \subseteq \mathbb{F}_2^n$ be linear code, $(a_0, \ldots, a_n)$ its weight distribution and $\lambda \in (0,1)$. For $0 \le i \le n$, let $i^* = \min\{i, n-i\}$. Then,*

$$a_i \;\le\; 2^{H(X|Y)} \cdot \begin{cases} \frac{|V|}{(1-\theta)^{i^*}(1+\theta)^{n-i^*}} & 0 \le i^* \le \frac{1-\theta}{2} \cdot n \\ \frac{|V|}{2^n} \cdot 2^{h_2(i/n)\cdot n} & \text{otherwise} \end{cases}$$

where X is a random uniform codeword in $V^\perp$, Y is the result of transmitting X over BEC(λ) *and $\theta = 2^\lambda - 1$.*

The proof of Proposition 1.3.15 uses simple Fourier analysis and is very similar to the proof of Proposition 1.6 in [Sam19]. We include a sketch to make the argument self-contained in Section 1.4.3.

Remark 6. *Since $2^{h_2(i/n)n} \le O(\sqrt{n})\binom{n}{i}$, whenever $H(X|Y) = o(n)$ for the dual code $V^\perp$, the weight distribution of the primal code V in a band of weights of width θ around $\frac{n}{2}$ is essentially upper-bounded by that of a random code of the same rate. This occurs even if $V^\perp$ does not achieve capacity: It is enough that $V^\perp$ decodes errors on* BEC(λ) *for some constant $\lambda < 1$ (cf. [KL97], where similar behavior was inferred for codes with large dual distance).*

In particular, consider a family of doubly transitive binary linear codes of constant rate R. Since by [KKM+17] such a family achieves capacity under bit-MAP decoding, due to (1.3) it will have the bound from Proposition 1.3.15 holding with $H(X|Y) = o(n)$. Such bound holds for both primal and dual codes, since the dual of a doubly transitive code is also doubly transitive.

Similarly, again building on [Sam19], we improve the bounds on the weight distribution of doubly transitive codes with distance $\Omega(n^\alpha)$.

Proposition 1.3.16. *Let $\{V_n\}$ be a family of doubly transitive binary linear codes of rate R and set $\alpha = \liminf_{n\to\infty} \frac{\log d(V)}{\log n}$. Also, let $(a_0, a_1, \ldots)$ denote the weight distribution of V_n. Then*

$$a_i \leqslant (2^{\lambda(R,\alpha)-o_n(1)} - 1)^{-i},$$

where $\lambda(R,\alpha)$ is as in Theorem 1.3.14.

One should compare the above with Proposition 1.6 in [Sam19]. The main difference is that Proposition 1.6 in [Sam19] applies only to codes that achieve capacity for the BEC under block-MAP decoding, and it is not known whether a doubly transitive code with minimum distance $\Omega(n^\alpha)$ for $\alpha < 1$ achieves capacity for the BEC under block-MAP decoding. Yet, Proposition 1.3.16 holds for any doubly transitive code with minimum distance $n^{\Omega(1)}$. Moreover, Proposition 1.6 in [Sam19] had an error term that dominated the estimate for weights $i = o(n)$.

For the weight distribution of Reed–Muller codes we obtain the following bound.

Proposition 1.3.17. *Let $(a_0, a_1, \dots,)$ denote the weight distribution of Reed–Muller codes with rate $R \in (0,1)$. Then*

$$a_i \leqslant O\Big((2^{1-R-o_n(1)} - 1)^{-i}\Big).$$

Again, the above estimate improves over [Sam19] by losing the error term that dominated the estimate for weights $i = o(n)$. However, for Reed–Muller codes there are stronger bounds for weights $i = n^{1-\delta}$ for every fixed constant $\delta \in (0,1)$ (e.g, see [KLP12, ASW15, KKM+16, SS20]). Hence, the improvement lies in the narrow region of weights $n^{1-o_n(1)}$ for some $o_n(1)$.

The proofs of Proposition 1.3.17 and Proposition 1.3.16 are omitted as those can be easily derived by the arguments of Theorem 1.3.14.

1.4 Appendix

1.4.1 Norms of Noisy Functions

In this section we prove Theorem 1.3.1. We start with the following lemma which is simply the formula for the rank of the dual matroid, but we state it as a separate claim because of its importance to us.

Lemma 1.4.1 (Proposition 2.1.9 in [Oxl06])**.** *Let V be a linear code of dimension k. Then,* $\dim V_S^\perp = |S| - \big(k - \dim V_{S^c}\big)$.

We now prove Theorem 1.3.1 by following the argument in [Sam19].

Proof of Theorem 1.3.1. Let V be a linear code of dimension k and $0 \le \lambda \le 1$. We shall prove that

$$\begin{aligned}\log \sum_{i=0}^{n} a_i (2^\lambda - 1)^i &\le \lambda n - \mathop{\mathbb{E}}_{S\sim\lambda} \dim V_S^\perp \qquad (1.8)\\ &= k - \mathop{\mathbb{E}}_{S\sim 1-\lambda} \dim V_S = H(X|Y)\ ,\end{aligned}$$

where X is a random uniform codeword in V and Y is the result of transmitting X over the channel BEC(λ). Define $f = 2^k 1_{C^\perp}$ and $\rho = \sqrt{2^\lambda - 1}$ and recall that the Walsh–Fourier transform of f satisfies $\widehat{f} = 1_C$. Since $\widehat{T_\rho f}(y) = \rho^{|y|}\widehat{f}(y)$, using Parseval's identity this means $\log \|T_\rho f\|_2^2 = \log \sum_{i=0}^n a_i (2^\lambda - 1)^i$. On the other hand, by properties of linear codes it can be checked that

$$\mathbb{E}(f|S)(x) = \begin{cases} 2^{|S| - \dim V_S^\perp} & \exists y \in V^\perp \text{ s.t. } x_S = y_S \\ 0 & \textit{otherwise} \end{cases},$$

and that $\Pr_x[\exists y \in C^\perp \text{ s.t. } x_S = y_S] = 2^{\dim V_S^\perp - |S|}$. Accordingly, $\log \|\mathbb{E}(f|S)\|_2^2 = |S| - \dim V_S^\perp$. The inequality in (1.8) follows by substituting on both sides of Theorem 1.2.17. We still need to justify the two equalities in (1.8). The first one follows immediately from Lemma 1.4.1. For the second equality, observe that $S \sim 1-\lambda$ has the same distribution as the set of non-erased coordinates over BEC(λ). Then, note that given $Y = y$ with non-erased coordinates in S, there are $2^{k-\dim V_S}$ equally likely possibilities for decoding, and therefore $H(X|Y=y) = k - \dim V_S$. □

1.4.2 The Bhattacharyya Bound

In this section we prove the well-known Bhattacharyya bound.

Proof of Theorem 1.3.3. We are analyzing the error probability of the block-MAP decoder for code V on the BMS channel $\mathcal{W}$. Since the code is linear and the channel symmetric, this is equal to the probability that the MAP decoder fails conditioned on the transmitted all-zeros codeword 0^n. Let Y be the output of the channel assuming the all-zeros codeword was transmitted.

We analyze the likelihood ratio between 0^n and another fixed codeword $x \in C$. Without loss of generality assume that $x = 1^i 0^{n-i}$ for some $0 < i \le n$. Assuming the decoder observes $y \in \mathcal{Y}^n$, the respective likelihood ratio is then[6]

$$\frac{\mathcal{W}(x|y)}{\mathcal{W}(0^n|y)} = \frac{\mathcal{W}(y|x)}{\mathcal{W}(y|0^n)} = \prod_{j=1}^{i} \frac{\mathcal{W}(y_j|1)}{\mathcal{W}(y_j|0)} \,.$$

Let $1 \le j \le i$ and define a random variable L_j equal to the likelihood ratio $\mathcal{W}(Y_j|1)/\mathcal{W}(Y_j|0)$ conditioned on all-zeros codeword. Recall the characterization of $\mathcal{W}$ as a mixture of BSC channels from Section 1.2, and let $Y_j = (P_j, X'_j)$. Observe that, conditioned on P_j, the likelihood ratio L_j is equal to $(1-P_j)/P_j$ with probability P_j and $P_j/(1-P_j)$ with probability $1-P_j$. Accordingly,

$$\mathbb{E}\left[\sqrt{L_j} \mid P_j\right] = 2\sqrt{P_j(1-P_j)} \,, \qquad\qquad \mathbb{E}\sqrt{L_j} = Z(\mathcal{W}) \,.$$

Let $\mathcal{E}(x)$ for $x \in C \setminus \{0^n\}$ denote the event $\frac{\mathcal{W}(x|y)}{\mathcal{W}(0^n|y)} \ge 1$ and let $P_B(x)$ be the probability of this event. Note that if the MAP decoder fails, then $\mathcal{E}(x)$ must have occurred for some x. By the considerations above and independence of L_j,

$$\begin{aligned} P_B(x) &\le \Pr\left[\prod_{j=1}^{i} L_j \ge 1\right] = \Pr\left[\prod_{j=1}^{i} \sqrt{L_j} \ge 1\right] \\ &\le \mathbb{E}\left[\sqrt{L_j}\right]^i = Z(\mathcal{W})^{|x|} \,. \end{aligned}$$

Finally, applying the union bound,

$$P_{\mathrm{B}}(\mathcal{W}, V) \le \sum_{x \in C, x \ne 0^n} P_B(x) \le \sum_{i=1}^{n} a_i Z(\mathcal{W})^i \,.$$ □

1.4.3 An Upper Bound on the Weight Distribution

Proof of Proposition 1.3.15. For this proof, let $f = 1_{V^\perp}$. In that case one checks that $\widehat{f} = \frac{1}{|V|} \cdot 1_V$. Furthermore, let $g(x) = \theta^{|x|}$ and verify that $\widehat{g}(y) = \frac{1}{2^n}(1-\theta)^{|y|}(1+\theta)^{n-|y|}$.

[6] We are abusing notation here, but the meaning of $\mathcal{W}(y|x)/\mathcal{W}(y|0^n)$ should be clear, at least for discrete channels and channels with distributions that have densities.

Let $(a_0, \dots, a_n)$, $(b_0, \dots, b_n)$ be the weight distributions of $V, V^\perp$ respectively. We calculate,

$$\frac{1}{2^n}\sum_{i=0}^{n} b_i\theta^i = \mathbb{E}_x f(x)g(x) = \sum_y \widehat{f}(y)\widehat{g}(y)$$
$$= \frac{1}{|V|}\cdot\frac{1}{2^n}\sum_{i=0}^{n} a_i(1-\theta)^i(1+\theta)^{n-i}$$

and consequently

$$\sum_{i=0}^{n} b_i\theta^i = \frac{1}{|V|}\cdot\sum_{i=0}^{n} a_i(1-\theta)^i(1+\theta)^{n-i} \,. \tag{1.9}$$

Substituting (1.9) into Theorem 1.3.1, we have for every $0 \le i \le n$

$$a_i \le 2^{H(X|Y)}\frac{|V|}{(1-\theta)^i(1+\theta)^{n-i}} \,.$$

This already establishes the result for $0 \le i < \frac{1-\theta}{2}n$. In fact, since the left-hand side of (1.1) is monotone in $\theta = 2^\lambda - 1$, we also have

$$a_i \le 2^{H(X|Y)}\frac{|V|}{(1-\alpha)^i(1+\alpha)^{n-i}}$$

for any $0 \le \alpha \le \theta$. If $\frac{1-\theta}{2}n \le i \le \frac{n}{2}$, then we take $\alpha = 1 - \frac{2i}{n} \le \theta$ and check that

$$\frac{1}{(1-\alpha)^i(1+\alpha)^{n-i}} = \frac{2^{h_2(i/n)n}}{2^n} \,,$$

therefore we have proved our statement for $0 \le i \le \frac{n}{2}$. To deal with the case $\frac{n}{2} < i \le n$, we invoke the calculation at the end of the proof of Proposition 1.6 in [Sam19] to see that

$$\sum_{i=0}^{n} a_{n-i}(1-\theta)^i(1+\theta)^{n-i} \le \sum_{i=0}^{n} a_i(1-\theta)^i(1+\theta)^{n-i}$$

and apply the argument above to a_{n-i} with $i \le \frac{n}{2}$. □

1.4.4 An Optimization Problem

We are interested in the following simple optimization problem for fixed $p, \beta \in (0,1)$,

$$\max_{1 \leqslant t \leqslant \frac{\beta}{p}} (p - \beta/t)^t$$

Let $f(t) = (p-\beta/t)^t = e^{\ln(p-\beta/t)t}$. Then we are looking for the global maximum of $f(t)$ in $[1, \infty)$. Differentiating $f(t)$ we get

$$f'(t) = e^{t\ln(p-\beta/t)}\cdot\left(\ln(p-\beta/t) + \frac{\beta/t}{p-\beta/t}\right) \tag{1.10}$$

The first term is positive so we focus on the term inside the parenthesis. Setting $s = p - \beta/t$ and equating the left term to zero we get the equation $\ln(s)s + p - s = 0$. The solution to this equation is $s_{opt} = e^{W_{-1}(-p/e)+1} = -\frac{p}{W_{-1}(-p/e)}$ where $W_{-1}(z)$ is the Lambert function. Note that the requirement $s \geqslant 0$, or equivalently $t \leqslant \frac{\beta}{p}$, is implicit if we take the real solution if this equation. Thus, $f(t)$ obtains its unique maximum, i.e $t_{opt} = \frac{\beta}{p - s_{opt}}$. Note that $t \ln(s) = -\beta/s$ and so in fact $f(t) = e^{-\beta/s}$ hence

$$f(t_{opt}) = e^{-\beta \cdot e^{-W_{-1}(-p/e)-1}} = \left(\frac{p/e}{-W_{-1}(p/e)}\right)^{\beta/p}$$

It remains to check when $t_{opt} \geqslant 1$ which happens precisely when $\beta \geqslant p + \frac{p}{W_{-1}(-p/e)}$. If $t_{opt} < 1$ then the solution to our optimization problem is simply attained at $t = 1$.

Chapter 2

Introduction to Space Bounded Derandomization

2.1 Computational Complexity

Computational complexity is the formal study of how resources affect computation. Most widely studied resources are *time* and *space* though there are many other well studied resources such as *randomness*, *communication*, *proof complexity* etc.

In theoretical computer science computation is often modeled using the celebrated model of a Turing machine (TM). Loosely speaking, a TM is an infinite tape on which it can read or write, and an algorithm is a finite sequence instructions for the TM. We should mention that there other models of computation, and indeed we shall define such later on (See Section 2.4).

2.2 Space Bounded Computation

The space complexity of an algorithm is the amount of *memory* it consumes, namely the length of tape needed for the Turing machine to execute the algorithm. We do not account for the input and output in the space complexity, hence the space complexity captures the amount of memory needed for the actual computation. This distinction is especially important when considering algorithms that use less space than their input or output length.

Formally, a deterministic space-bounded algorithm is a Turing machine which 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 [AB09b, Chapter 4] and [Gol08, Chapter 5].

Definition 2.2.1. *We define* $\mathbf{DSPACE}(S)$ *as the class of problems that can be solved via an algorithm that uses* $O(S)$ *space, and set* $\mathbf{L} \overset{\text{def}}{=} \mathbf{DSPACE}(\log n)$.

The class $\mathbf{L}$ is arguably the lowest studied complexity class of Turing machines. For instance, $\mathbf{L} \subseteq \mathbf{P}$ contained in the class of problems that can be solved in polynomial time. Still, it is a major open problem whether $\mathbf{L} \overset{?}{\neq} \mathbf{P}$, and in fact we do not know to separate it from stronger classes e.g if $\mathbf{L} \overset{?}{=} \mathbf{PH}$[1]. Unfortunately, proving such statements is out of reach.

In order to "get the feel" of space-bounded algorithms, we review some of the basic operations that can be carried out efficiently in terms of space. It is easy to see, that computing $x+y$, where x, y are two n-digit integers can be done $\log n$ space via the elementary long addition. Less obvious, but also not too difficult, is that computing the iterated addition

$$x_1 + x_2 + \cdots + x_n$$

where $x_1, \ldots, x_n$ are all k-digit integers requires $O(\log n + \log k)$ space. This also uses long addition, and hinges on the observation that the carry never exceeds n (and so can be stored via $\log n$ bits). As a consequence, we get that computing the product

$$x \cdot y$$

where x, y are two n-digit integers can be done $O(\log n)$ space via the elementary long multiplication, which reduces to iterated addition. Lastly, we remark that the iterated product

$$x_1 \cdots x_n,$$

and division $\lfloor \frac{x}{y} \rfloor$ can also be computed in logarithmic space, but this is highly non-trivial [CDL99, HAB02]. In the Boolean world, it is possible to evaluate *Boolean Formulas* in logarithmic space. To put it briefly, a Boolean formula over the variables $\{x_1, \ldots, x_n\}$ is a directed a-cyclic graph in which every vertex has at most one outgoing edge, and every vertex is either labeled with a variable x_i or an elementary Boolean $\{\wedge, \vee, \neg\}$ (AND/OR/NEGATION). The evaluation over a specific input $(a_1, \ldots, a_n) \in \{True, False\}^n$ is obtained by placing the Boolean value of a_i on vertices labeled by x_i, and propagating Boolean values according to the labels written on the vertices. The output is then written on the vertices that has no out-going edge. For further details see [Gol08, Chapter 1.2.4].

What about complex operations such as matrix multiplication? It is tempting to suggest that using the building blocks of addition, and multiplication one can derive a space-efficient algorithm for computing matrix products - this is correct, but in a very subtle way. The algorithmic principle behind this, often taken as granted, is that computational building blocks can be assembled at the cost of their total cost. While this is trivial in the context of time-bounded algorithms, it is much less obvious for space-bounded algorithms since **intermediate computations cannot be stored for free**. Nonetheless, the following theorem suggests that one can space-efficiently *compose* algorithms.

Claim 2.2.2 (Composition of Space-Bounded Algorithms (e.g., [Gol08, Lemma 5.2])). *Let*

$$f_1, f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$$

be two 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

$$O(s_1(\ell_2(n)) + s_2(n) + \log(n + \ell_1(n))),$$

where $\ell_1(n), \ell_2(n)$ are bounds on the output length of f_1, f_2 on inputs of length n.

[1]The class **PH** contains **NP** and its complement **coNP**. It can be defined using oracle TMs as follows. Define recursively $\Sigma_0 = \mathbf{P}$, $\Sigma_i = \mathbf{NP}^{\Sigma_{i-1}}$ then $\mathbf{PH} = \cup_{i=0}^{\mathbb{N}} \Sigma_i$. For more information see Chapter 3 in [Gol08].

Corollary 2.2.3. *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*

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

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

For more information on space efficient algorithmic composition the reader may consult [Gol08, Chapter 5.1]. While Claim 2.2.2 is not a triviality both result-wise and proof-wise, one should keep in mind that it is by no mean the optimal approach. For instance, computing the n-th power of an n-digit integer by composing the squaring function $f(x) = x^2$ logarithmically many times yields the sub-optimal space complexity of $O(\log^2 n)$.

Using the space composition theorem (Claim 2.2.2) we can now give a space-efficient algorithm for computing matrix products.

Definition 2.2.4 (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 $|A| = \Theta(kw^2)$.*

Claim 2.2.5. *The function $f(A,B) = AB$ can be computed in space $O(\log |A| + \log |B|)$.*

Proof. First, consider the function $g(A,B,i,j) = (A[i,\ell] \cdot B[\ell,j])_{\ell=1}^{w}$, which can be computed in logarithmic space by iterating over ℓ, and computing the product. Composing g with the iterated function yields the matrix product function $h(A,B,i,j) = AB[i,j]$. □

Claim 2.2.6. *The matrix powering function $f(A,n) = A^n$ can be computed in space $O(\log^2 n + \log n \cdot \log |A|)$.*

Proof. Composing $g(A,A)$ for k times, we can compute A^{2^k} in space

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

following corollary 2.2.3, and Claim 2.2.5. (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. □

Claim 2.2.7. *The iterated matrix product function $f(A_1, A_2, \ldots, A_n) = A_1 \cdots A_n$ can be computed in space $O(\log^2 n + \log n \cdot \log \max |A_i|)$.*

Proof. Similarly, compose the function

$$f((A_j, B_j)_{j=1}^{s}) = (A_j \cdot B_j)_{j=1}^{s}$$

$\log n$ times. □

There is also an approximated version of matrix products, in which we output an approximation to the matrix product. The idea is very simple - always truncate all but the $\log \frac{n}{\varepsilon}$ most significant digits.

Claim 2.2.8. *There exists an algorithm that takes as input $A \in \mathbb{R}^{w \times w}$ and output a matrix A such that*

$$\forall i, j \in [w] \quad A[i,j] - A^n[i,j] \leqslant \varepsilon,$$

and runs in $O(\log n \log\log \frac{n}{\varepsilon} + \log n \log \max |A_i|)$ space.

What about approximating the iterated product

$$A_1 \cdots A_n \ ?$$

The same holds, but there is a subtlety, and it seems that the standard approach using the composition theorem Claim 2.2.2 induces an extra $\log^2 n$ in the space complexity. In a nutshell, if one define function

$$f((A_j, B_j)_{j=1}^s) = \lfloor (A_j \cdot B_j)_{j=1}^s \rfloor_{\log \frac{n}{\varepsilon}}$$[2]

analogously to the exact iterated product, then indeed

$$|f \circ f \circ \cdots \circ f(A_1, \ldots, A_n)[i,j] - (A_1 \cdots A_n)[i,j]| \leqslant \varepsilon.$$

The problem is that in most invocations of f the output length is $\Omega(n)$, and there are $\log n$ compositions, hence Claim 2.2.2 yields an extra term of $\log^2 n$. The solution follows from a natural generalization of the space-bounded composition (Claim 2.2.2) outlined in Section 5.7.

2.3 Randomized Computation

Randomized algorithms are algorithms that use randomness, usually independent unbiased coin flips, as part of their execution. A randomized algorithm is allowed to be incorrect with some probability over its internal coins as long as its failure probability is not too large. We emphasize that randomized algorithms are required to be correct on every input with high probability (w.h.p).

To the uninitiated, randomized algorithms might seem a little bit dubious. Nonetheless, evidently randomness is a very powerful tool in designing algorithms. Randomized algorithms tend to be extremely simple and efficient, and are widely used both in practice and in theory. We give two instructive examples:

- Primality Testing: Primality testing is the problem of deciding whether a given integer (in binary representation) is prime or not. While polynomial time randomized algorithms were known since the 70's [SS77, Rab80], the goal of designing a deterministic algorithm remained elusive. It took thirty years until Agrawal, Kayal, and Saxena [AKS04] gave a deterministic algorithm for primality testing which is more complicated than previous randomized algorithms both in terms of implementation and running time.

- Polynomial Identity Testing (PIT): The problem of deciding whether a given polynomial is identically zero[3]. This problem has a very simple randomized algorithm though no deterministic algorithm is known.

[2]The notation $\lfloor \cdot \rfloor_t$ means the entry-wise truncation of all but the first t bits.

[3]Obviously, we are not given their coefficients. The standard definition is that the input is an arithmetic circuit which allows us to efficiently compute values of the polynomial but its coefficients. For further details see [SY10].

By design, randomized algorithms come at the expense of being occasionally wrong. We distinguish between three different types of error:

1. Two-Sided Error: The algorithm is allowed to be incorrect in cases that the answer is "Yes" and also in cases that the answer is "No".
2. One-Sided Error: In cases that the answer is "No" the algorithm has to be correct (i.e, with probability 1), and in cases the answer is "Yes" the algorithm may be incorrect with small probability. There is an analogous definition with the roles of "Yes" and "No" reversed.
3. Zero-Sided Error: The algorithm outputs the correct answer with large probability, but it is allowed to say "Don't Know" (it never outputs an incorrect answer).

Definition 2.3.1. *Let $S, R\colon \mathbb{N} \to \mathbb{N}$ increasing, and let n denote the input length for the algorithm.*

- *Define* **BPL** *to be the class of problems that can be solved via a randomized two-sided error algorithm that uses $O(\log n)$.*
- *Define* **RL** *to be the class of problems that can be solved via a randomized one-sided error algorithm that uses $O(\log n)$.*
- *Define* **ZPL** *to be the class of problems that can be solved via a randomized zero-sided error algorithm that uses $O(\log n)$.*

We emphasize that random coins are not exempted from the space complexity. That is, if we wish to store previous coin tosses we have to store these as part of the computation. Formally, the coin tosses are written on a unidirectional tape which means that we can only have access to the the current coin toss (and not previous ones).

There is subtlety that arises in the context of randomized space-bounded computation: does the algorithm is allowed not to halt? The above definitions refer to algorithms that always halt. A more elaborated discussion regarding the halting issue is addressed in Chapter 4.

2.4 Branching Programs

It is standard to replace the computational model of space bounded TMs with the model of Read-Once Branching Programs (ROPBs). A regular, labeled, layered graph over alphabet Σ is a directed graph in which every vertex has $|\Sigma|$ out-going edges each labeled by a distinct symbol from Σ. We shall use the following terminology.

- The number of layers minus one is called the *length*, and is denoted by n.
- The maximum number of vertices in every layer is called the *width*, and is denoted by w. Hence, every vertex is identified by its layer and an index $i \in [w]$ called *state*.
- Every vertex has $|\Sigma|$ outgoing edges each labeled with a distinct symbol from Σ.

See Figure 2.1 for illustration.

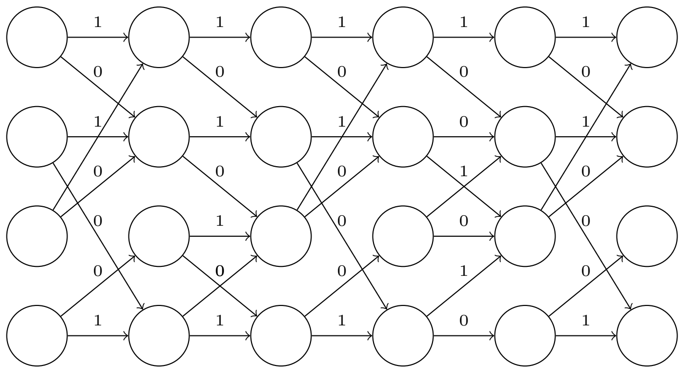


Figure 2.1: An $(n = 5, w = 4, \Sigma = \{0, 1\})$ ROBP

It is more convenient a regular, labeled, layered graph via its neighbor function: Let $B = (B_1, \ldots, B_n) \in ([w] \times \Sigma \to [w])^n$ be a sequence of functions

$$B_i \colon [w] \times \Sigma \to [w],$$

then the regular, labeled, layered graph $G_B = ([n+1] \times [w], E_B)$

$$E_B \overset{\text{def}}{=} \{((i, w), (i + 1, B_i(w, \sigma))) \mid \sigma \in \Sigma,\ i \in \{1, \ldots, n\}\} \tag{2.1}$$

defines a layered graph of length n, width w over alphabet Σ. Conversely, every regular, labeled, layered graph defines such sequence of functions. Also, for $1 \leqslant i < j \leqslant n$ we define the sub-graph of $B \in ([w] \times \Sigma \to [w])$ by

$$B_{i,j} = (B_i, B_{i+1}, \ldots, B_j).$$

Definition 2.4.1. *An ROBP of length n, width w, over alphabet Σ is a triplet comprising a regular, labeled, layered graph B, a start state v_0, and a subset of accepting states $V_{acc} \subseteq [w]$*

$$(B, v_0, V_{acc}) \in ([w] \times \Sigma \to [w])^n \times [w] \times \mathcal{P}([w]).$$

We shall say that n is the length of the ROBP, w its length, and Σ is its alphabet.

2.4.1 Computation

Given an ROBP (B, v_0, V_{acc}) we define the computation of an ROBP as follows: Given a sequence of symbols $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma^n$ we set

$$B(v_0, \sigma) \overset{\text{def}}{=} B_n(\cdots B_3(B_2(B_1(v_0, \sigma_1), \sigma_2)\sigma_3) \cdots, \sigma_n) \in [w].$$

In words, the ROBP starts at state v_0, moves to state $v_1 = B_1(v_0, \sigma_1)$, moves to state $v_2 = B_2(v_1, \sigma_2)$ etc. Pictorially, interpreting the ROBP as a layered graph, the computation follows the labels $\sigma_1, \sigma_2, \ldots, \sigma_n$ starting from v_0 at the first layer, and outputs the state it reaches in the last layer.

Furthermore, given a subset of the vertices V_{acc} we can define the notion of acceptance: if $B(v_0, \sigma) \in V_{acc}$ then we say that the ROBP B accepts σ (with respect to the starting state v_0, and accept states V_{acc}). We remark that usually ROBPs are defined with a fixed initial state v_0, and accept states V_{acc} but we shall not use this terminology.

For example, consider the above layered graph in Figure 2.1, and take v_0 the upper vertex in the first layer (i.e., state 1), and accepting states $V_{acc} = \{1, 2, 4\}$. Then, the computation over this ROBP with input $\sigma = (1, 0, 0, 1, 1)$ reaches state 1 - which is an accepting state so the ROBP accept.

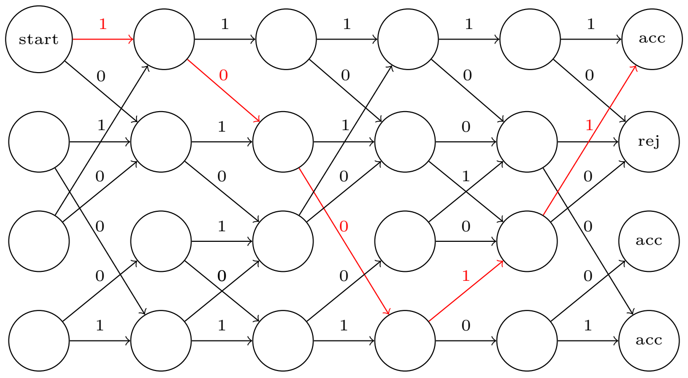


Figure 2.2: An $(n = 5, w = 4, \Sigma = \{0, 1\})$ ROBP

2.4.2 Correspondence To Randomized Space Bounded Algorithms

We now explain why ROBPs are a good substitute for the computational model of randomized space bounded TMs. A configuration of an algorithm is a snapshot consisting of the algorithm's current storage and state. The configuration graph is the digraph of all configurations with edges going between consecutive configurations namely, the edges are all pairs (C_1, C_2) where C_2 is the configuration obtained by running the algorithm a single step from configuration C_1. Note that a randomized algorithm essentially takes two inputs: The "true" input, and the random coins. If we take an algorithm A, and an input x then its execution is a function of the random coins which can be computed by a ROBP. We give the precise analogy: Given a randomized algorithm $A(x, y)$ with "true" input x and random coins $y \in \Sigma^n$.

- The ROBP is the layered graph, where in each layer we have all possible configurations and edges going between consecutive configurations, v_0 is the starting configuration, and V_{acc} the accepting configurations of the algorithm.
- The input for the ROBP are the random coins for the algorithm.
- The space of the algorithm is then roughly $\log(w)$ since w vertices corresponds to $\log(w)$ bits of information[4].
- The probability that the ROBP accept a random input equals the acceptance probability of x by the algorithm.

2.4.3 The Transition Matrix

Let $B \in ([w] \times \Sigma \to [w])^n$. Each B_i where $B_\ell(\sigma)$ corresponds to $|\Sigma|$ Boolean matrices of dimension $w \times w$

$$B_\ell(\sigma)[i, j] = \begin{cases} 1 & B_\ell(i, \sigma) = j \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

and a single $w \times w$ matrix

$$\mathsf{M}(B_\ell) \overset{\text{def}}{=} \mathbb{E}_\sigma[B_\ell(\sigma)]. \tag{2.3}$$

[4]Here we say roughly because one should also account for the states of the algorithm etc.

Alternatively, the i,j-th entry of $\mathsf{M}(B_\ell)$ is the probability of moving from state i to state j upon reading a random symbol from Σ. The definitions of $B_\ell(\sigma)$ and $\mathsf{M}(B_\ell)$ naturally extends to B by matrix multiplication

$$B(\sigma) \overset{\text{def}}{=} B_1(\sigma_1) \cdots B_1(\sigma_n), \quad \mathsf{M}(B) \overset{\text{def}}{=} \mathsf{M}(B_1) \cdots \mathsf{M}(B_n). \tag{2.4}$$

Observe that $B(\sigma)$, $\mathsf{M}(B)$ still carry the same semantics: the i,j-th entry of $B(\sigma)$ is 1 iff starting from state i moves to state j upon reading σ, and the i,j-th entry of $\mathsf{M}(B)$ is the probability of moving from state i to j upon reading a random input. Therefore, each row of the transition matrix has nonnegative entries which sum to one - such matrices are called *stochastic matrices.*

Definition 2.4.2. *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*, and sub-stochastic if its entries are nonnegative and every row sums to at most* 1*.*

An immediate implication of the above observation is that the existence of an algorithm that approximates powers of stochastic matrices that runs in logarithmic space implies

$$\mathbf{BPL} = \mathbf{L}.$$

To some extent, the problem of approximating stochastic matrices captures the class **BPL** (See Section 4.1.

Remark 7. *There is a well known unfortunate notional issue of transition matrices. We chose to denote $A[i,j]$ for the transition from state i to j, and so the matrix multiplication goes right to left in Equation (2.4). It is very common that the initial state is a distribution $\pi \in \mathbb{R}^w$, rather than a fixed state, and so by applying the ROBP to that distribution we get another distribution over the states which equals*

$$\pi A.$$

For the uninitiated, it might be slightly annoying, or inconvenient to multiply by a vector from the left. The alternative, which is very common and respectable, it to define $A[i,j]$ for the transition from state j to i, and then we have $A\pi$ instead of the above. Since the notation πA will not appear in this work, we chose to use the first option.

Recall from our perspective, an ROBP stands for simulating a randomized algorithm on a specific input, and the inputs for the ROBP are the random coins. In this setting, the transition matrix encompasses all the "computational information" as it accounts for the acceptance probability. Therefore it is natural to measure *closeness* between two ROBPs by measuring the distance between their transition matrices (i.e., with respect to some norm).

There are numerous norms in the literature for measuring matrices in the context of ROBP, but this work features the two most basic ones: the maximum norm, and the induced ℓ_∞ norm.

Definition 2.4.3. *For a matrix $A \in \mathbb{R}^{w \times w}$:*

1. *Maximum Norm:* $\|A\|_{\max} = \max_{i,j\in[w]} |A[i,j]|$.
2. *Infinity Norm:* $\|A\|_\infty = \max_{i\in[w]} \sum_{j\in[w]} |A[i,j]|$, .

There is a very natural interpretation of the induced ℓ_∞ norm $\|\cdot\|_\infty$. Suppose that $M_1, M_2 \in \mathbb{R}^{w\times w}$ are the transition matrices of $B^{(2)}, B^{(1)} \in ([w] \times \Sigma \to [w])^n$. By standard arguments

$$\|M_1 - M_2\|_\infty = 2 \cdot \max_{v_0\in[w], S\subseteq[w]} \left| \Pr_\sigma[B^{(1)}(v_0,\sigma) \in S] - \Pr_\sigma[B^{(2)}(v_0,\sigma) \in S] \right|.$$

In fact, by convexity this is the same as taking the maximum over any initial distribution π supported on $[w]$. Thus, the induced infinity norm between two transition matrices of two ROBPs equals the largest difference in acceptance probability going over all possible initial states v_0, and possible accepting states V_{acc}.

Remark 8. *Generally, for two discrete distributions π_1, π_2 supported on Ω the quantity*

$$\max_{S \subseteq \Omega} \left| \Pr_{X \sim \pi_1}[X \in S] - \Pr_{X \sim \pi_2}[X \in S] \right|$$

is known as the statistical distance *between π_1, π_2 or* total variation distance. *It is well-known that the statistical distance between two distributions also equals their ℓ_1 distance (as vectors).*

We finish with two useful and simple inequalities. The first inequality, relates the maximum norm, and the induced ℓ_∞ norm.

Claim 2.4.4. *For any matrix $M \in \mathbb{R}^{w \times w}$ we have that*

$$\|M\|_{\max} \leqslant \|M\|_\infty \leqslant w\|M\|_{\max}.$$

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

Claim 2.4.5. *Let for any $A_1, \dots, A_k, B_1, \dots, B_k$ satisfying $\|A_i\|_\infty, \|B_i\|_\infty \leqslant 1$*

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

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

2.5 Pseudorandomness and Derandomization

Understanding the power of randomness as a computational resource is a major problem in complexity. In particular, whether randomness can be eliminated or reduced in some cases. This process of taking a randomized algorithm and reducing the amount of randomness it uses, ideally converting it into a deterministic algorithm, is called *derandomization*. Perhaps the most famous open problem in this context is whether any polynomial time randomized algorithm can be derandomized. This is often referred to as the $\mathbf{BPP} \stackrel{?}{=} \mathbf{P}$ problem. An amazing line of work showed that this problem is intimately related to circuit lower bounds. Specifically, sufficiently strong circuit lower bounds will settle the $\mathbf{BPP} \stackrel{?}{=} \mathbf{P}$ conjecture [NW94, IW94, BFNW93], and conversely the $\mathbf{BPP} \stackrel{?}{=} \mathbf{P}$ conjecture implies circuit lowers bounds [KI04] (though slightly weaker bounds). Unfortunately, proving circuit lower bounds is considered to be notoriously difficult which puts the lid on efforts to resolve the $\mathbf{BPP} \stackrel{?}{=} \mathbf{P}$ conjecture any time soon. Still, dernadomiztion of specific problems such as the PIT problem is an active and flourish area of research.

There are a few concrete methods to derandomize algorithms, which are applicable in certain situation, though we are interested in derandomizing all space bounded algorithms. This means that we are looking for a more methodological derandomization technique. We shall roughly distinguish between two types of derandomization:

- Black-Box: Derandomization which depends only on the algorithm's functionality (i.e., whether it accepts or rejects certain inputs).

- White-Box: Derandomization which depends on the algorithm's instructions.

The simplest example of white-Box derandomization for space bounded computation is to compute the exact acceptance probability using matrix multiplication which takes $O(\log^2(n))$ space (See Section 2.4 for the relation to matrix multiplication). A less naive example is a result by Reingold-Trevisan-Vadhan [RTV06] who showed that any randomized space bounded algorithm with one-sided error can be transformed to a connectivity problem of a digraph with a *certain property*. Therefore, solving the connectivity problem on digraphs with that property in log-space yields $\mathbf{RL} = \mathbf{L}$.

The following subsection deals with *pseudorandom generators* (PRGs), which are the ultimate means of black-box derandomization.

2.5.1 Pseudorandom Generators

Intuitively speaking, a pseudorandom generator uses *few* truly random coins and outputs a sequence of *pseudorandom* coins. The hope is that these pseudorandom coins can be used of truly random coins in a given algorithm with the hope that it works *roughly the same*. Thus, the algorithm works just the same, but with fewer random coins hence dernadomizing the algorithm.

A typical way of getting a deterministic algorithm from PRGs is by running the algorithm on all possible inputs for the PRG and taking the majority vote. It is ludicrous to suggest that such an object can possibly work for all algorithms, so it is necessary to restrict the class of algorithms that the PRG is useful against. Note that this derandomization technique is oblivious to the functionality of the algorithm we want to derandomize. Traditionally, a PRG against a class of Boolean functions $\mathcal{C} \subseteq \Sigma^n \to \{0,1\}$ is a function $G\colon \{0,1\}^s \to \Sigma^n$ satisfying

$$\forall f \in \mathcal{C} \quad |\mathbb{E}_\sigma[f(\sigma)] - \mathbb{E}_z[G(z)]| \leqslant \varepsilon. \tag{2.5}$$

The parameter s is called the *seed length*, z is called the seed, and ε is called the *accuracy* or *error* of the PRG. It is common to say that G ε-fools the class $\mathcal{C}$ (though we will not be using this terminology).

In our setting, the class $\mathcal{C}$ is the class of ROBPs of length n, width w, over alphabet Σ, namely $\mathcal{C} = ([w] \times \Sigma \to [w])^n$. There is a technical issue: Equation (2.5) does not compile as $\mathbb{E}_\sigma[B(\sigma)]$ is a matrix, rather than a number (see Section 2.4 for definitions). Instead we require that

$$\forall B \in [w] \times \Sigma \to [w] \quad \|\mathbb{E}_\sigma[B(\sigma)] - \mathbb{E}_z[G(z)]\|_\infty \leqslant \varepsilon. \tag{2.6}$$

Following the remark at the end of Section 2.4.3, one can verify that this the above is actually equivalent had we included the starting and accepting states in the definition of ROBPs so that $B(\sigma) \in \{0,1\}$, and used the definition in Equation (2.5) instead. Regardless of how ROBPs are defined, all the PRGs against ROBPs that appear in the literature are analyzed are analyzed by showing that Equation (2.6) holds. Moreover, Equation (2.6) may be relaxed to allowing arbitrary matrix norms yielding a much richer definition. E.g., [HPV21] constructed a PRG against a certain class of ROBPs in the induced ℓ_2 norm (a.k.a spectral norm or operator norm).

It is well known that (non-explicit) pseudorandom generators exists with essentially optimal seed length. In fact, a random function is with high probability a PRG against any class of functions which is not too large.

Claim 2.5.1. *Let $\mathcal{C} \subseteq \{0,1\}^n \to \{0,1\}$ and $\varepsilon > 0$ then there exists a function $G\colon \{0,1\}^s$ such that Equation (2.5) holds with $s = \log\log|\mathcal{C}| + \log\frac{1}{\varepsilon} + O(1)$.*

Noting that $|([w] \times \Sigma \to [w])^n| = w^{nw|\Sigma|}$ we get the following corollary[5].

Corollary 2.5.2. *Let $\varepsilon > 0$ then there exists a function $G\colon \{0,1\}^s$ such that Equation (2.6) holds with $s = O\big(\log \frac{nw}{\varepsilon}\big)$.*

We now define PRGs against the model of ROBP.

Definition 2.5.3. *An (n, w, Σ, ϵ) PRG with seed length s is a function $G\colon \{0,1\}^s \to \Sigma^n$ such that for every (n, w, Σ) ROBP B*

$$\big|\mathbb{E}_{x\sim\Sigma^n}[B(x)] - \mathbb{E}_{z\sim\{0,1\}^s}\big|B(G(z)) \leqslant \epsilon$$

In the above definition we considered the Accept/Reject definition, though we can analogously define PRGs against ROBPs in the matrix formulation.

Definition 2.5.4. *Let $\|\cdot\|$ be a matrix norm[6]. An $(n, w, \Sigma, \epsilon, \|\cdot\|)$ PRG with seed length s is a function $G : \{0,1\}^s \to \Sigma^n$ such that for every (n, w, Σ) ROBP B*

$$\big\|\mathbb{E}_{x\sim\Sigma^n}[B(x)] - \mathbb{E}_{z\sim\{0,1\}^s}[B(G(z))]\big\| \leqslant \epsilon \tag{2.7}$$

Note that in this setting $B(x)$ is a matrix.

Proposition 2.5.5. *There exists a non-explicit (n, w, Σ, ϵ) PRG with seed length $s = O(\log(nw|\Sigma|/\epsilon))$.*

Proposition 2.5.6. *If there exists an (n, w, Σ, ϵ) PRG with seed length $s = O(\log(nw|\Sigma|/\epsilon))$ that runs in space $O(s)$ then* **BPL** $=$ **L**.

2.6 Brief Account of Space Bounded Derandomization

Understanding the role that randomness plays in computation is of central importance in complexity theory. While randomness is provably necessary in many computational settings such as cryptography, PCPs and distributed computing, it is widely believed that randomness adds no significant computational power to neither time- nor space-bounded algorithms. Remarkably, proving such a statement for time-bounded algorithms implies circuit lower bounds which seem to be out of reach of current proof techniques [NW94, IKW02, KI04].

On the other hand, there is no known barrier for proving such a statement in the space-bounded setting. Indeed, while we cannot even rule out a scenario in which randomness "buys" exponential time, the space-bounded setting is much better understood. Savitch's theorem [Sav70] already implies that any one-sided error randomized algorithm can be simulated deterministically with only a quadratic overhead in space, namely $\mathbf{RL} \subseteq \mathbf{L}^2$. The (possibly) stronger inclusion $\mathbf{BPL} \subseteq \mathbf{L}^2$ can be proven easily through a variant of Savitch's theorem and also follows from [BCP83]. Using pseudorandom generators, Nisan [Nis92, Nis94] devised a time-efficient derandomization with quadratic overhead in space, concretely, $\mathbf{BPL} \subseteq \mathbf{DTISP}(\mathrm{poly}(n), \log^2 n)$. Focusing solely on space, the state of the art result was obtained by Saks and Zhou [SZ99] that build on Nisan's work to deterministically simulate two-sided error space s randomized algorithms in space $O(s^{3/2})$, thus, establishing that $\mathbf{BPL} \subseteq \mathbf{L}^{3/2}$.

[5]Here it is actually useful to consider the setting in which an ROBP is defined with an initial vertex and accepting states so that the function $B(\sigma)$ is Boolean.

[6]One may also consider notions of distance which are not a norm.

2.7 Efficient Implementations of PRGs Against ROBPs

We now discuss two fundamental constructions of PRGs against the model of ROBP: the INW ([INW94]) and Nisan ([Nis92]) generators, and present specific space-efficient implementations for those. The constructions and their analysis are well known, except for the space complexity which is implicit in those works and also depends on the specific implementation (the variant of INW we use was explored by [HV06]).

2.7.1 Nisan's Generator

Nisan, in his seminal work [Nis92], constructed a family of pseudorandom generators against the class of ROBPs of length n, width w over alphabet Σ with accuracy ε using seed of length

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

We briefly recall the construction and its properties. Without the loss of generality, $|\Sigma| = \Theta\left(\frac{nw}{\varepsilon}\right)$[7], and let

$$\mathcal{H} \subseteq \Sigma \to \Sigma$$

with be a two-universal family of hash functions of size $|\mathcal{H}| = |\Sigma|^2$. The seed for

$$\mathsf{N} \stackrel{\text{def}}{=} G_{\log n}\colon \{0,1\}^d \to \Sigma^n$$

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 $s = O(\log n \cdot \log |\Sigma|)$. We define $G_i\colon \Sigma \times \{0,1\}^{i \cdot 2 \log |\Sigma|} \to \Sigma^{(2^i)}$ recursively as follows

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

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

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

recalling that $\mathsf{M}(B) = \mathbb{E}_{z \in \Sigma^n}[B(z)]$. Moreover, N can be computed in

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

space.

[7]If the alphabet is small then we can enlarge it by collapsing layers into a single one hence increasing the alphabet.

The analysis can be found at [Nis92], though the space complexity is glossed over so we give a rough sketch. In order to analyze the space complexity one has to consider a specific implementation of the hashing family $\mathcal{H}$. A possible choice is

$$\mathcal{H} = \{h_{a,b}(x) = ax + b \mid a, b \in \mathbb{F}_{2^k}\},$$

where $|\Sigma| = 2^k$. This calls for a space-efficient implementation of arithmetic over the finite field $\mathbb{F}_{2^k}$ which can be found in e.g. [HAB02, HV06]. Given $\sigma \in \Sigma$,$h = (h_1, \dots, h_{\log n}) \in \mathcal{H}^{\log n}$, and an index $j = (b_1, \dots, b_{\log n}) \in \{0,1\}^{\log n}$ we can compute the j-th output symbol of $\mathsf{N}(\sigma, h)$ in the following two alternative ways:

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

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

space.

- Alternatively, we can do the above computation using composition of space bounded functions (Claim 2.2.2), resulting in space complexity

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

2.7.2 The INW Generator

We consider the INW generator [INW94] instantiated with *seeded-extractors* (as, e.g., in [RR99])[8]. Again, as we are not going to discuss the correctness of the construction, the definition of seeded-extractors, and for that matter other primitives as well are irrelevant for this part.

We are given parameters n, Σ, w, and ε. Given functions

$$\mathsf{Ext}_i \colon \{0,1\}^{m+id} \times \{0,1\}^d \to \{0,1\}^{m+id}$$

we define recursively

$$G_i(x \circ y) = G_{i-1}(x) \circ G_{i-1}(\mathsf{Ext}(x,y)),$$

where $G_0(x) \stackrel{\text{def}}{=} x$, and we set the INW generator $\mathsf{INW} \stackrel{\text{def}}{=} G_{\log n}$.

Theorem 2.7.2 ([INW94, RR99]). *Suppose that $\mathsf{Ext}_i \colon \{0,1\}^{m+id} \times \{0,1\}^d \to \{0,1\}^{m+id}$ is a $(m + id - \Delta, \varepsilon_{\mathsf{Ext}})$ seeded-extractor then for every ROBP $B \in ([w] \times \{0,1\}^m \to [w])^n$*

$$\|\mathbb{E}[B(\sigma)] - \mathbb{E}[B(\mathsf{INW}(z)]\|_\infty \leqslant n(\varepsilon_{\mathsf{Ext}} + w2^{-\Delta}).$$

Setting, $\varepsilon_{\mathsf{Ext}} = \frac{\varepsilon}{2n}$, and $\Delta = \log \frac{2nw}{\varepsilon}$ we get accuracy ε, and the seed length is given by $m + d\log n$. Moreover, in the setting $\varepsilon_{\mathsf{Ext}} = \frac{\varepsilon}{2n}$, and $\Delta = \log \frac{2nw}{\varepsilon}$ we can implement the generator in

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

space where $\Sigma = \{0,1\}^m$.

[8]The connection between randomness extractors and derandomization of space bounded computation originated in [NZ96].

For further details the reader may consult [AB09a, Chapter 16]. It follows that to implement the INW generator we need a space-efficient *seeded extractor* with a *small entropy loss* in the *high min-entropy regime*. Goldreich and Wigderson [GW97] gave such a construction utilizing a regular expander $H = (V, E)$ with a small normalized second eigenvalue. For our expander, we choose a Cayley graph over the commutative group $\mathbb{Z}_2^n$ with a generator set $S \subseteq \{0,1\}^n$ that is λ-biased. It is well known that $Cay(\mathbb{Z}_2^n, S)$ has normalized second largest eigenvalue at most λ. For the λ-biased set we choose a construction from [AGHP92]. Altogether, this unfolds for the following.

- For the λ-biased set S, first pick q to be the first power of two larger than $\frac{n}{\lambda}$. The set S is of cardinality q^2. For every $\alpha, \beta \in \mathbb{F}_q$ there is an elements $s_{\alpha,\beta} \in \mathbb{Z}_2^n$ where $(s_{\alpha,\beta})_i = \langle \alpha^i, \beta \rangle$, such that multiplication is in $\mathbb{F}_q$ and the inner product is over $\mathbb{Z}_2$. [AGHP92] showed the set is λ-biased.

- We let $H = (V, E)$ with $V = \mathbb{Z}_2^n$ and $(x, y) \in E$ iff $x + y \in S$. H is a λ-expander.

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

Claim 2.7.3. *Let $0 < \Delta < n$ and set H and GW as above. Then, $\mathsf{GW}\colon \{0,1\}^n \times [D] \to \{0,1\}^n$ is a $(k = n - \Delta, \varepsilon)$ extractor with seed length $d = O(\Delta + \log \frac{n}{\varepsilon})$ and space complexity $O(\log n \cdot \log(\Delta + \log(n/\varepsilon)))$.*

The expander mixing lemma basically shows that GW is an $(n - \Delta, \varepsilon = O(2^{\Delta/2}\lambda))$ extractor, and the seed length of this extractor is

$$\log |S| = O(\log \frac{n}{\lambda}) = O(\log \frac{n2^{\Delta}}{\varepsilon}) = O(\Delta + \log \frac{n}{\varepsilon}).$$

Again, the uninitiated reader is referred to [AB09a, Chapter 16]. The space complexity of computing $\mathsf{GW}(x, y)$ given x and y, is the space needed to compute $s_y \in S$ from $y = (\alpha, \beta) \in \mathbb{F}_q^2$, plus the space needed to compute $x + s_y$. The dominating step in computing s_y is computing α^i (for $i \le n$) which can be done in $O(\log n + \log\log q)$ using space-efficient arithmetic over $\mathbb{F}_{2^k}$ (e.g., [HAB02]). Altogether, the space needed is

$$O(\log n + \log\log \frac{n}{\lambda}) = O\left(\log n + \log \Delta + \log\log \frac{1}{\varepsilon}\right).$$

We note that Healy and Viola [HV06] gave an extremely efficient implementation of the above AGHP generator, yielding a better space complexity of $O(\log(n + \log q))$ to compute $\langle \alpha^i, \beta \rangle$. However, in our overall setting of parameters it will make negligible difference. We remark that by using expanders with better dependence between D and λ, one can get $d = O(\Delta + \log \frac{1}{\varepsilon})$, but here we care more about the space complexity, and $\log n$ factors are negligible for us.

The previous passage concludes the implementation of the seeded extractor needed for the INW generator, which we can now implement. Given a seed $x \in \{0,1\}^s$ we view the computation of $\mathsf{INW}(z)$ as a full binary tree of depth $\log n$. Given an index $j \in [n]$, computing $\mathsf{INW}(x)_j \in \{0,1\}^m$ can be done by walking down the computation tree, and each time either truncating a string or invoking an extractor. By composition of space-bounded functions (Claim 2.2.2) the space complexity of the construction is $\log n$ times the space complexity of the worst extractor used. That is,

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

Chapter 3

Error Reduction For Weighted PRGs Against ROBPs

3.1 Background

3.1.1 Pseudorandom Pseudo-Distributions for ROBPs

In [BCG20] Braverman, Cohen and Garg introduced the notion of a *pseudorandom pseudo-distribution* (PRPD) generalizing pseudorandom distributions. A PRPD is a distribution

$$\widetilde{\mathcal{D}} = ((\rho_1, \sigma_1), \dots, (\rho_{2^s}, \sigma_{2^s}))$$

where $\rho_1, \dots, \rho_{2^s} \in \mathbb{R}$ and $\sigma_1, \dots, \sigma_{2^s} \in \Sigma^n$. They constructed a PRPD with the property that for every ROBP $B \in ([w] \times \Sigma \to [w])^n$

$$\left\| \left| \sum_i \rho_i B(\sigma_i) \right| - \mathbb{E}_\sigma[B(\sigma)] \right\|_\infty \leqslant \varepsilon.$$

Note that the definition of a PRPD allows the weights ρ_i to take both positive and negative values. These values are not necessarily bounded by 1 in absolute value, nor by any constant for that matter, and they do not necessarily sum up to 1. Nevertheless, the definition requires that the numbers cancel out nicely so that summing the weights of the respective paths and, in particular, the sum is a number in $[-\varepsilon, 1+\varepsilon]$. Analogous to a PRG, a *weighted pseudorandom generator* (WRPG) is a function

$$G \times \mu \colon \{0,1\}^s \to (\Sigma \times \mathbb{R})^n$$

whose output, when fed with a uniform seed, is a PRPD. Similarly to a PRG, we say that $G_{\mathsf{R}} \times \mu$ is WPRG against the class $\mathrm{B}[n, w, \Sigma]$ with accuracy ε if for every $B \in ([w] \times \Sigma \to [w])^n$

$$\| |\mathbb{E}_z[\mu(z) B(G(z))|] - \mathbb{E}_\sigma[B(\sigma)] \|_\infty \leqslant \varepsilon.$$

Observe that if $\mu \equiv 1$ then the notion of a WPRG coincides with that of a PRG.

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

3.1.2 The Error Parameter

Braverman et al. [BCG20] constructed a WPRG that has seed length with an improved, in fact near-optimal, dependence on the error parameter ε. Their WPRG has seed length

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

For the purpose of derandomization, the error parameter is anyhow taken to be constant, and so the necessity of such an improvement may seem moot. However, by inspecting Nisan's recursive construction one can see that the $\log^2 n$ term in the seed length appears due to the way the error evolves throughout the recursion. Hence, a construction which allows for a more delicate error analysis is called for. Furthermore, the Saks–Zhou construction applies Nisan's PRG in a setting in which $\varepsilon \ll 1/n$ for obtaining their result. It was observed [BCG20] that improving upon [SZ99] can be obtained by constructing a PRG having seed length with better dependence on both w, ε, even when retaining the $\log^2 n$ dependence.

Interestingly (and unfortunately), the $\log^2 n$ term in the BCG construction appears for a completely different reason. In short, unlike prior works [Nis92, INW94] that maintain a list of instructions throughout the recursion, BCG maintains a more involved structure consisting of several lists of lists. Maintaining the invariant on this complex structure is the reason for the $\log^2 n$ term in the seed of BCG's construction.

As hinted above, the BCG construction is quite involved. In a subsequent work Chattopadhyay and Liao [CL20] somewhat simplified the BCG construction also obtaining slight improvement in parameters. In particular, the seed length obtained by [CL20] is

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

Additionally, Hoza and Zuckerman [HZ20] obtained a significantly simpler construction of hitting sets against ROBPs. Their construction has seed length

$$O\left(\frac{1}{\max(1, \log\log w - \log\log n)} \cdot \log n \cdot \log nw + \log \frac{1}{\varepsilon}\right).$$

Although hitting sets are weaker objects than PRPDs that are aimed for the derandomization of one sided error randomized algorithms, a subsequent work by Cheng and Hoza [CH20] showed how to derandomize two sided error randomized algorithms using hitting sets.

3.2 Our Result

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

Theorem 3.2.1 ([CDR⁺21])**.** *Let $G\colon \{0,1\}^{s_G} \to \Sigma$ a PRG against* $\mathrm{B}[n, w, \Sigma]$ *with accuracy*

$$\|\mathbb{E}_x[B(G(x))] - \mathbb{E}_\sigma[B(\sigma)]\|_\infty \leqslant \varepsilon_G,$$

where we assume that $\varepsilon_G \leqslant \frac{1}{4(n+1)}$. Then, there exists a WPRG $G_{\mathsf{R}} \times \mu\colon \{0,1\}^{s_{\mathsf{R}}} \to \Sigma$ *against* $\mathrm{B}[n, w, \Sigma]$ *with accuracy*

$$\|\mathbb{E}_x[\mu(x) B(G_{\mathsf{R}}(x))] - \mathbb{E}_\sigma[B(\sigma)]\|_\infty \leqslant \varepsilon.$$

Moreover,

$$s_{\mathsf{R}} = s_G + \log|\Sigma| + O(\log\frac{w}{\varepsilon}\log\log\frac{1}{\varepsilon}),$$

and if G requires S space then G_{R} takes $O(S + \log\log\frac{w}{\varepsilon}\log\log^2\frac{1}{\varepsilon})$ space.

When instantiated with Nisan's PRG [Nis92] our error reduction procedure yields WPRGs with a seed that is slightly shorter than [BCG20] and is incomparable to [CL20].

Corollary 3.2.2. *There exists a WPRG against* $\mathrm{B}[n, w, \Sigma]$ *with accuracy ε, and seed length*

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

computable in space $O\big(\log(nw|\Sigma|) + \log\log\frac{1}{\varepsilon}\log\log\frac{w}{\varepsilon}\big)$.

Note that for ε which is not tiny the space complexity is dominated by the first term. Specifically, for

$$\varepsilon > 2^{-2^{\log^{1/3} n}}, \quad w < 2^{2^{\log^{1/3} n}}$$

the space complexity is indeed $O(\log(nw|\Sigma|))$. Had we used INW instead [INW94] (Theorem 2.7.2, the space complexity would deteriorate to

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

Our error reduction procedure as well as the resulting WPRG are significantly simpler than that of [BCG20, CL20]. Moreover, the underlying ideas are different and conceptually cleaner. More generally, it is much preferred to have a black-box error reduction procedure rather than a specific explicit construction. On top of the insights obtained, such a modularization has the potential of being instantiated in different settings such as for regular and permutation ROBPs or for bounded-width ROBPs.

Our error reduction procedure borrows ideas from the line of work concerning deterministic space-efficient graph algorithms, in particular a recent work by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford and Vadhan [AKM+20] (which, in turn, is based on an exciting line of work on nearly-linear time graph algorithms, deterministic or otherwise. See [CKP+16, CKP+17] and references therein).

Independently, Pyne and Vadhan [PV21] also used the Richardson iteration to obtain a WPRG for polynomial-width branching programs, and furthermore used that to obtain new results for permutation ROBPs.

3.3 Overview

Let $B \in ([w] \times \Sigma \to [w])^n$ with transition matrices

$$M_1, M_2, \dots, M_n$$

that is, $M_i = \mathbb{E}_\sigma[B_i(\sigma)]$, and let $G\colon \{0,1\}^{s_G} \to \{0,1\}^n$ be a PRG against $\mathrm{B}[n, w, \Sigma]$ with accuracy ε_G which we wish to increase. Following [AKM+20] we observe that by denoting

$$L[i,j] \overset{\text{def}}{=} \begin{cases} -M_i & j = i+1 \\ I & i = j \\ 0 & \textit{otherwise} \end{cases}$$

one has that,

$$L^{-1}[i,j] \overset{\text{def}}{=} \begin{cases} M_{i,j-1} & i \leqslant j \\ 0 & \textit{otherwise} \end{cases} .$$

In words, L is a block-matrix in which the main diagonal is the identity matrix, the diagonal above it has $-M_i$ on it, and the remaining blocks are zero. Its inverse L^{-1}, is an upper-triangular block-matrix such that the a,b-th block is the product

$$M_a \cdot M_{a+1} \cdots M_{b-1}.$$

Richardson iteration is a method for improving a given approximation to an inverse of a matrix that is frequently used to construct a preconditioner to a Laplacian system. To describe this method, let L be a matrix and suppose that B is an approximation for L^{-1}. For $k \geqslant 1$ define the matrix

$$\mathsf{R} = \sum_{i=0}^{k} (I - BL)^i B. \tag{3.1}$$

It can be shown that

$$\left\|\mathsf{R} - L^{-1}\right\| \leq \left\|L^{-1}\right\| \cdot (\|L\| \cdot \left\|B - L^{-1}\right\|)^{k+1}.$$

In our setting, the matrix L can be approximated via the PRG G as follows. Define the $(n+1)w \times (n+1)w$ lower triangular block matrix $\widetilde{M}$ as follows. For $a, b \in [n+1]$, $a < b$, and $\sigma \in \{0,1\}^s$, let

$$\widetilde{M}[a,b] = \mathbb{E}_x[B(G(x)_{[i,j]})].$$

Further, $\widetilde{M}[a,a] = I_w$. Since G has error ε_G, one can easily verify that

$$\left\|\widetilde{M}[a,b] - M_a \cdots M_b\right\|_\infty \leqslant (n+1)\varepsilon_G.$$

Equivalently, the PRG G can be used to mildly approximate L^{-1} by applying it to all sub-programs of the original ROBP. Roughly, by taking

$$B = \widetilde{M}, \quad \varepsilon_G = n^{-2}, \quad k = O(\log_n \frac{1}{\varepsilon}),$$

one obtains the approximation

$$\left\|\mathsf{R} - L^{-1}\right\|_\infty \leqslant \varepsilon. \tag{3.2}$$

In particular, the upper-right block of R approximates of the desired product $M_1 \cdots M_n$ within accuracy ε.

We further develop Equation (3.1). Let $\Delta = I - BL$. It is possible to show that

$$\Delta[a,b] = \begin{cases} \widetilde{M}[a,b-1] \cdot M_{b-1} - \widetilde{M}[a,b] & a < b, \\ 0 & a \geqslant b. \end{cases} \tag{3.3}$$

Substituting this back to R, for $a < b$ we have that

$$\mathsf{R}[a,b] \;=\; \sum_{m=0}^{k} \sum_{a<r_1<\cdots<r_i\leqslant b} \Delta[a,r_1] \cdot \Delta[r_1,r_2] \cdots \Delta[r_{i-1},r_i] \cdot \widetilde{M}[r_1,b].$$

If we further let $C_0[a,b] = \widetilde{M}[a,b-1] \cdot M_b$ and $C_1[a,b] = \widetilde{M}[a,b]$ then

$$\mathsf{R}[a,b] = \sum_{m=0}^{k-1} \sum_{\substack{a<r_1<\cdots<r_m\leqslant b \\ t_1,\ldots,t_m \in \{0,1\}}} (-1)^{t_1+\cdots+t_m} C_{t_1}[a,r_1] C_{t_2}[r_1,r_2] \cdots C_{t_m}[r_{m-1},r_m] C_1[r_m,b]. \tag{3.4}$$

Recall that each block in $\widetilde{M}[a,b]$ corresponds to simulating B over the segment $[a,b]$ using the generator $G\colon \{0,1\}^s \to \Sigma^n$ (truncated to its first $b-a$ symbols). Hence, every summand in Equation (3.4)

$$C_{t_1}[a,r_1] \cdots C_{t_i}[r_{i-1},r_i] \cdot \widetilde{M}[r_i,b]$$

can be realized as the transition matrix of an ROBP of length $n_{\text{aux}} = i+1$, of the same width w, and over a larger alphabet. Specifically, the alert reader can see that this alphabet can be taken to be

$$\Sigma_{aux} = \{0,1\}^s \times \Sigma.$$

Our construction thus uses an auxiliary PRG against $\mathsf{B}[n_{aux}, w, \Sigma_{aux}]$ with accuracy ε_{aux} and hence approximates each summand to accuracy ε_{aux}. Therefore, replacing the seeds for G by the output of the auxiliary PRG approximates R to within accuracy

$$\varepsilon_{aux} n^{O(k)} \approx \varepsilon_{aux} \cdot \varepsilon^{-\Omega(1)},$$

which in turn approximates L^{-1} to accuracy ε (see Equation (3.2)) yielding an overall approximation of accuracy $O(\varepsilon)$. As the ROBP that correspond to each summand is short, recall

$$i \leqslant k = O\left(\log_n \frac{1}{\varepsilon}\right) \ll n,$$

a short seed is required even for the high accuracy $\varepsilon_{aux} = \varepsilon^{O(1)}$ that we require. We invoke [INW94] as our auxiliary PRG as it has good dependence on the alphabet size which, in our case, is comparable to the seed of the crude PRG that we started with. We remark that the weights in our WPRG $G_{\mathsf{R}} \times \mu$ are used to realize Equation (3.4) as the expectation

$$(3.4) \approx \mathbb{E}_z[\mu(z) B(G_{\mathsf{R}}(z))].$$

Indeed, on top of the sign, one has to account for this averaging by taking the weights to be large as the number of summands in Equation (3.4), so that the expectation becomes a sum.

3.3.1 Comparison with [BCG20]

It is worthwhile to explore the differences between the BCG construction [BCG20] (and the followup work of Chattopadhyay and Liao [CL20] which uses similar ideas) and ours and to point out the aspects of our work that we find similar to the work of Cheng and Hoza [CH20], and of Hoza and Zuckerman [HZ20]. We start by giving a brief overview of the BCG construction.

A brief overview of BCG

In constructions prior to [BCG20] (e.g., [Nis92, INW94]), a list of instructions is maintained with the property that given a ROBP $A_1, \ldots, A_n$, averaging over the products corresponding to the instructions yields the desired approximation to the product $A_n \cdots A_1$. The key idea suggested in [BCG20] is to maintain not a single list whose average yields the desired approximation but rather several lists of instructions $L_0, L_1, \ldots, L_k$ such that averaging according to the instructions in L_0 yields a modest approximation; averaging according to $L_0 \cup L_1$ yields a more refined approximation, and so forth. Averaging according to the instructions given by $L_0 \cup \cdots \cup L_k$ gives the desired approximation. Thus, L_0 can be thought of as a crude approximation, L_1 a first order correction term, L_2 a second order correction term, etc.

To implement this idea, weights were introduced and, moreover, each list but for L_0 was in itself a list of lists, or bundles. The different instructions in a bundle did not carry useful information by themselves and it is the bundle which has the desired properties. Lists that correspond to higher error terms requires the expensive use of bigger bundles and larger weights, and so a delicate use of balanced and unbalanced samplers is employed in [BCG20] in order to maintain the desired invariant throughout the recursion and assuring that the bundles and weights do not get too large.

Comparison with [BCG20]

Our work, in comparison, goes back to the use of a single list as in [Nis92, INW94]. We do not need to maintain several lists, let alone lists of bundles. This makes our construction significantly simpler and, in particular, spares us from the delicate application of different types of samplers. The only component we do need are weights, both positive and negative that are unbounded in absolute value. However, it is straightforward to pinpoint the weights used by our construction whereas in [BCG20] the weights are computed via a recursive algorithm. As a result, it is difficult to argue about them. We believe that the simpler and more explicit structure of our construction would enable future works to combine our construction with other ideas for the purpose of obtaining improved constructions and derandomization results.

The common theme to both our construction and BCG is working with cancellations. We "read of" Richardson iteration what cancellations to consider. As we discussed in the end of Section 3.3, we interpret Richardson iteration as comparing a PRG with the PRG obtained by replacing the first bit by a fresh truly random bit. The BCG construction, on the other hand, "plant" cancellations by considering two samplers–one more refined than the other–and encode their difference in their lists (this requires the introduction of bundles). So, in a sense, BCG's cancellations are obtained by comparing one approximation to another where both approximations are obtained via samplers whereas we make use of one approximation coming from a PRG and another that is obtained by replacing the first bit by a fresh truly uniform bit. The way we combine these is dictated by Richardson iteration.

Common aspects with [HZ20, CH20]

For their derandomization result, Cheng and Hoza [CH20] introduce the notion of *local consistency*. Informally, the authors consider the difference between applying a generated sequence of instructions (via a hitting set) to that obtained by the generated sequence when replacing the last bit with a fresh truly random bit. This is somewhat reminisce to the way we read the cancellations of the Richardson iteration. However, while local consistency is used for making decisions once a ROBP is given, we combine the analog sequences using the Richardson iterator in a block-box matter.

The construction of Hoza and Zuckerman [HZ20] also shares similar aspects with ours. There, they start with a modest-error PRG to get an ε-error hitting set by running the PRG for $k = \log_n(1/\varepsilon)$ times according to partitions of $[n]$ to k segments, resembling what we do. Instead of drawing the PRG's seeds uniformly at random, they derandomize the construction using a hitter. We note however, that their analysis is very different from ours, and uses a progress measure concerning reaching an accepting state.

3.4 Preliminaries

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

Definition 3.4.1 (WPRG)**.** *We say $G \times \mu \colon \{0,1\}^s \to (\Sigma \times \mathbb{R})^n$ is a WPRG against $\mathrm{B}[n,w,\Sigma]$ with accuracy ε if for every ROBP $B \in ([w] \times \Sigma \to [w])^n$*

$$\|\mathbb{E}_z[\mu(z)B(G(z))] - \mathbb{E}[B(\sigma)]\|_\infty \leqslant \varepsilon.$$

Also, s is called the seed length of $G \times \mu$.

3.5 The Richardson Iteration

Let A be an invertible $n \times n$ real matrix, and assume that B approximates A^{-1}, concretely,

$$\left\|B - A^{-1}\right\| \leqslant \varepsilon_0$$

for some sub-multiplicative norm. Richardson iteration is a method for obtaining a more refined approximation of A^{-1} given access to the crude B as well as to the original matrix A.

Lemma 3.5.1. *Let $L \in \mathbb{R}^{m \times m}$ be an invertible matrix and $A \in \mathbb{R}^{m \times m}$ such that $\left\|L^{-1} - A\right\| \le \varepsilon_0$. For any nonnegative integer k, define*

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

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

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

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

In particular, for $Z = AL$,

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

Thus,

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

□

Following [AKM⁺20] we will be interested in the following instantiation of the Richardson iteration. Let $M = (M_1, \ldots, M_n)$ be a sequence of $w \times w$ matrices. Following [AKM⁺20] we consider the $(n+1)w \times (n+1)w$ matrix

$$M = \begin{pmatrix} 0 & M_1 & \ldots & 0 & 0 \\ 0 & 0 & M_2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ \vdots & & & \ddots & M_n \\ 0 & 0 & \ldots & \ldots & 0 \end{pmatrix}. \tag{3.5}$$

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

Lemma 3.5.2. *For $i, j \in [n+1]$, the (i,j)-th block of L^{-1} is given by*

$$L^{-1}[a,b] = \begin{cases} M_a \cdots M_{b-1} & a < b, \\ I_w & a = b, \\ 0 & b > b. \end{cases}$$

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

$$\mathsf{M}(B) \stackrel{\text{def}}{=} M_1 \cdots M_n,$$

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

Lemma 3.5.3. *Let $B = (B_1, \ldots, B_n) \in ([w] \times \Sigma \to [w])^n$. Set $M_i = \mathbb{E}_{\sigma \in \Sigma}[B_i(\sigma)]$ and L as in Equation (3.5). Also, let $G \colon \{0,1\}^s \to \Sigma^n$ be a PRG against $\mathrm{B}[n, w, \Sigma]$ with accuracy ε_G, and consider*

$$\widetilde{M}[a,b] \stackrel{\text{def}}{=} \begin{cases} \mathbb{E}_{x \in \{0,1\}^s}\left[B_{[a,b-1]}\big(G(x)_{[a,b]}\big)\right] & a \leqslant b \text{ [1]} \\ 0 & a > b \end{cases} \tag{3.6}$$

Then,

$$\left\| L^{-1} - \mathsf{R}(\widetilde{M}, L, k) \right\| \le (n+1) \cdot ((2n+2))\varepsilon_G)^{k+1}.$$

3.6 Richardson Iteration Is a WPRG

So far, we have seen that the Richardson iteration can be used to increase the accuracy of any given approximation to the iterated product

$$\mathsf{R}(\widetilde{M}, L, k)[1, n+1] \approx M_1 \cdots M_n.$$

From an algorithmic standpoint, using efficient matrix multiplication and addition, one can efficiently compute the matrix $\mathsf{R}(\widetilde{M}, L, k)$. However, we further claim that the Richardson iteration can be written as a WPRG.

Lemma 3.6.1. *Let $L, \widetilde{M}$ as in Section 3.5. Then, there exists an explicit function $G' \times \mu_{\mathsf{R}} \colon \{0,1\}^{s'} \to \Sigma^n \times \mathbb{R}$ such that*

$$\mathsf{R}(\widetilde{M}, L, k) = \mathbb{E}_z\left[\mu_{\mathsf{R}}(z) G'(z)\right].$$

[1]The convention is that $\widetilde{M}_{i,i} = I$.

Proof. Let $\widetilde{M}$ as in Equation (3.6) and write

$$\mathsf{R}(\widetilde{M}, L, k) = \sum_{i=0}^{k} \Delta^i \widetilde{M}$$

where

$$\Delta \overset{\text{def}}{=} I - \widetilde{M}(I - M) = \widetilde{M}M - (\widetilde{M} - I).$$

Note that $L \overset{\text{def}}{=} I - M$ is zero except on the diagonal above the main diagonal, namely $L[a, b] = 0$ for $b \neq a + 1$. Also, $L[a, a+1] = -\widetilde{M}[a, a+1]$[2] so it is not hard to verify that

$$\Delta[a, b] = \begin{cases} \widetilde{M}[a, b-1]M_{b-1} - \widetilde{M}[a, b] & a < b \\ 0 & a \geqslant b \end{cases}.$$

In this notation $\Delta[a, b]$ is a matrix where $a, b \in \{1, \ldots, n\}$, and so Δ vanishes below the main diagonal. We shall denote the two matrices in Δ by C_0 and C_1

$$C_0[a, b] = \widetilde{M}[a, b-1]M_{b-1} \ , C_1[a, b] = \widetilde{M}[a, b]. \tag{3.7}$$

Plugging everything back to the Richardson iteration we get the following formula

$$\mathsf{R}[a, b] = \sum_{m=0}^{k-1} \sum_{\substack{a<r_1<\cdots<r_m\leqslant b \\ t_1,\ldots,t_m\in\{0,1\}}} (-1)^{t_1+\cdots+t_m} C_{t_1}[a, r_1]C_{t_2}[r_1, r_2]\cdots C_{t_m}[r_{m-1}, r_m]C_1[r_m, b], \tag{3.8}$$

with the convention that the single term corresponding to $m = 0$ is $\widetilde{M}[a, b]$. Again, the above is a sum of matrices, and so $\mathsf{R}[a, b]$ denotes a matrix. It is time to try interpret the above sum in terms of the PRG G. Recall that for $a < b$

$$M[a, a+1] = \mathbb{E}_\sigma[B_a(\sigma)], \ \widetilde{M}[a, b] = -\mathbb{E}_x[B_a(G(x)_{[a,b-1]})]$$

and so substituting this into C_0 and C_1 yields

$$C_0[a, b] = \mathbb{E}_{x,\sigma}[B_{[a,b-1]}(G(x)_{[a,b-2]} \circ \sigma)], \tag{3.9}$$
$$C_1[a, b] = \mathbb{E}_x[B_{[a,b-1]}(G(x)_{[a,b-1]})]. \tag{3.10}$$

In other words, C_1 simulates the PRG G on B, and C_0 does the same but the last symbol is "fresh", i.e. drawn uniformly and independently. Also, $C_0[a, b]$, $C_1[a, b]$ are by themselves a sum over all x, σ. Denote

$$C_0[a, b](x, \sigma) \overset{\text{def}}{=} B_{[a,b-1]}(G(x)_{[a,b-2]} \circ \sigma), \tag{3.11}$$
$$C_1[a, b](x, \sigma) \overset{\text{def}}{=} B_{[a,b-1]}(G(x)_{[a,b-1]})] \tag{3.12}$$

so that

$$C_0[a, b](x, \sigma) = \mathbb{E}_{x,\sigma}[C_0[a, b](x, \sigma)], \tag{3.13}$$
$$C_1[a, b](x, \sigma) = \mathbb{E}_{x,\sigma}[C_1[a, b](x, \sigma)] \tag{3.14}$$

[2]Assuming that the PRG is marginally uniform which is always the case.

Note that $C_1[a,b]$ does not depend on σ but we include it anyway. Also note that $C_0[a,b](x,\sigma)$, and $C_1[a,b](x,\sigma)$ are sequence of symbols of the appropriate length namely

$$C_0[a,b](x,\sigma),\ C_1[a,b](x,\sigma) \in \Sigma^*$$

in contrast to $C_0[a,b], C_1[a,b]$ which are matrices. For concreteness, let us focus on the entry $\mathsf{R}[1,n+1]$, which approximates the transition matrix of B and so we can re-write Equation (3.8) as

$$\mathsf{R}[1,n+1] = \mathbb{E}_{m,t,r,x,\sigma}\left[\mu(m,t)B(G'(m,t,r,x,\sigma))\right], \tag{3.15}$$

where

$$(m,t,r,x,\sigma) = (m,(t_1,\dots,t_m),(r_1,\dots,r_m),(x_1,\dots,x_{m+1}),(\sigma_1,\dots,\sigma_m)) \tag{3.16}$$

$$G'(m,t,r,x,\sigma) = C_{t_1}[1,r_1](x_1,\sigma_1)\cdots C_{t_m}[r_{m-1},r_m](x_m,\sigma_i)\widetilde{M}[r_m,n](x_{m+1}), \tag{3.17}$$

How should we define μ? First, μ has to account for the signs

$$(-1)^{t_1+\dots+t_m},$$

but that is not all. The problem is that Equation (3.15) is written as a sum, while a PRG is the expectation (or average) over its seeds. To correct that, one has to account for this averaging by taking the weights to be large which cancel out and become a sum. Specifically, we simply set the magnitude of μ to be the number of summands in Equation (3.15)

$$\mu(m,t) = \left(\sum_{m=0}^{k-1}\binom{n-1}{m}2^m\right)\cdot(-1)^{t_1+\dots+t_m}. \tag{3.18}$$

The reason that x-s, and σ-s are not taken into account in μ is that we already take the expectation over those, so their weights do not need correction. In conclusion, Equation (3.15) shows that the Richardson iteration is in fact WPRG. $\square$

3.7 The Construction

The problem with the Richardson iteration as a WPRG is that it is too costly in terms of randomness. Let us now account for the seed length s', namely the amount of randomness needed to simulate the Richardson iteration as a WPRG:

i. Choose a partition of the interval $1,2,\dots,n$ into i sub-intervals $[1,r_1]$, $[r_1,r_2]$, $\dots$, $[r_m,n]$. It is not hard to see that there are
$$\sum_{m=0}^{k-1}\binom{n-1}{m}$$
such partitions.

ii. Simulate the execution of B over each interval using the PRG G **independently**, namely with independent seeds.

iii. Refresh Symbols: For every interval $[r_m,r_{m+1}]$ choose whether to add a "refresh" symbol in which the replace the last symbol of the PRG, with a uniform random symbol from Σ.

iv. Signs: If the number of non-refreshed intervals (excluding the last interval) is odd then we add a minus sign (excluding the last interval which is always without a refresh symbol).

In total, if we also include the refresh symbols, the signs, the seeds required to simulate the PRG on every interval then this amounts to at most

$$k(\log|\Sigma| + s_G) + \log\left(\sum_{m=0}^{k-1} \binom{n-1}{m} 2^m\right)$$

many bits. The reason for the "at most" is that we should include a refresh symbol only if $t_m = 0$, though it will not make much of a difference. Also, we could try to use a shorter seed when simulating G on short intervals, but again this does not lead to better parameters. Still, Equation (3.15) falls short of proving Theorem 3.2.1 because it is inefficient in terms of the randomness complexity. To see that, note that in order to get an error ε, one has to take k to be roughly

$$k = \log\frac{1}{\varepsilon},$$

though this results in a seed length which is around $s_G \cdot \log\frac{1}{\varepsilon}$. In contrast, Theorem 3.2.1 gives an additive term of $\log\frac{1}{\varepsilon}$. In order to improve upon the seed length, we do not choose the seeds for every application of G independently, but rather pseudorandomly using an auxiliary PRG. Fix a specific summand in Equation (3.15)

$$B(C_{t_1}[1,r_1](x_1,\sigma_1)\cdots C_{t_m}[r_{m-1},r_m](x_m,\sigma_i)\widetilde{M}[r_m,n](x_{m+1})).$$

and let $(x_1,\sigma_1),\ldots,(x_{m+1},\sigma_{m+1})$ be free variables (σ_{i+1} is not used). The above summand can be viewed as an ROBP

$$B_j^{(m,t,r)}(x_j,\sigma_j) \overset{\text{def}}{=} \begin{cases} B_{[r_j,b]}(x_j) & j=m \\ B_{[r_j,r_{j+1}-2]}(x_j)\circ B_{r_{j+1}-1}(\sigma_j) & t_j=0,\ j<m \\ B_{[r_j,r_{j+1}]-1}(x_{j+1}) & t_j=1,\ j<m \end{cases}.$$

It follows that

$$B^{(m,t,r)} \overset{\text{def}}{=} (B_1^{(m,t,r)},\ldots,B_k^{(m,t,r)})$$

has length $n_{aux} = k$, width $w_{aux} = w$ (same as B), over the alphabet $\Sigma_{aux} = \{0,1\}^{s_G}\times\Sigma$. Hence, it is fooled by any PRG against the corresponding class of ROBPs. Let

$$G_{aux}\colon \{0,1\}^{s_{aux}} \to \Sigma_{aux}^k$$

be the INW generator of Theorem 2.7.2 with accuracy ε_{aux}, and seed length

$$s_{aux} = s_G + \log|\Sigma| + O\left(\log^2 k + \log\frac{kw}{\varepsilon_{aux}}\right).$$

The reason for using the INW generator is its additive dependence on $\log|\Sigma|$. Had we used Nisan's PRG from Theorem 2.7.1 instead of INW then the seed length of the original PRG G would deteriorate by a factor of $\log\log\frac{1}{\varepsilon}$.

We are ready to define the generator, and prove Theorem 3.2.1. Define

$$G_{\mathsf{R}}(z,(m,t,r)) \overset{\text{def}}{=} G'(m,t,r,G_{aux}(z)), \tag{3.19}$$

where $0 \leqslant m \leqslant k-1$, $G_{aux}(z) = (x_1\circ\sigma_1,\ldots,x_{m+1}\circ\sigma_{m+1})$ (unused values of G_{aux} are ignored).

3.8 Proof of Correctness

Proof of Theorem 3.2.1. We shall prove that G_{R} as in Equation (3.19) satisfies the requirements of Theorem 3.2.1, with

$$k = \frac{\log \frac{2(n+1)}{\varepsilon}}{\log \frac{1}{2(n+1)\varepsilon_G}}, \text{ and } \varepsilon_{aux} = \frac{\varepsilon^3}{4(n+1)^2}.$$

Let $B \in ([w] \times \Sigma \to [w])^n$, and $\varepsilon_G \leqslant \frac{1}{4(n+1)}$. For every fixed $i \in \{0, 1, \ldots, k-1\}$, partition $r = (r_1, \ldots, r_m)$, and signs $t = (t_1, \ldots, t_m)$ we have that

$$\left\| \mathbb{E}_z[B^{(m,t,r)}(G_{aux}(z))] - \mathbb{E}_\alpha[B^{(m,t,r)}(\alpha)] \right\|_\infty \leqslant \varepsilon_{aux},$$

and so by Equation (3.15)

$$\left\| \mathbb{E}\left[\mu(m,t) G'(m,t,r,G_{aux}(z))\right] - \mathsf{R}[1,n+1] \right\|_\infty \leqslant \left(\sum_{m=0}^{k-1} \binom{n-1}{m} 2^m \right) \cdot \varepsilon_{aux}.$$

By Lemma 3.5.3

$$\|\mathbb{E}_\sigma[B(\sigma)] - \mathsf{R}[1,n+1]\|_\infty \leqslant (n+1)(2\varepsilon_G)^k,$$

and so

$$\left\| \mathbb{E}\left[\mu(m,t) G'(m,t,r,G_{aux}(z))\right] - \mathsf{R}[1,n+1] \right\|_\infty \leqslant \left(\sum_{m=0}^{k-1} \binom{n-1}{m} 2^m \right) \cdot \varepsilon_{aux} + (n+1) \cdot (2(n+1)\varepsilon_G)^k.$$

By our choice of k we have that

$$(n+1) \cdot (2(n+1)\varepsilon_G)^k = \varepsilon/2.$$

As for the other term, we have the trivial bound

$$\sum_{m=0}^{k-1} \binom{n-1}{m} 2^m \leqslant n^{2k} = \frac{4(n+1)^2}{\varepsilon^2}.$$

By our choice of ε_{aux} we have that $\frac{4(n+1)^2}{\varepsilon^2} \cdot \varepsilon_{aux} \leqslant \varepsilon$ which establishes

$$\|\mathbb{E}[\mu(m,t) B(G_{\mathsf{R}}(z,(m,t,r)))] - \mathbb{E}_\sigma[B(\sigma)]\|_\infty \leqslant \varepsilon.$$

We are left with the space complexity which follows from the space complexity required to compute the INW generator Theorem 2.7.2, and space composition theorem (Claim 2.2.2). $\square$

In order to derive Corollary 3.2.2 we instantiate Theorem 3.2.1 with G being the Nisan's PRG from Theorem 2.7.1.

Chapter 4

Approximating Powers of Stochastic Matrices in Small Space

4.1 Background

4.1.1 The Landscape of Randomized Space-Bounded Algorithms

In his survey on space-bounded derandomization [Sak96], Michael Saks proposed a notional system for classifying space-bounded randomized algorithms. Inspired by [BCD+89], he identified three important characteristics:

i. Zero/One/Two-Sided Error (See Section 2.3).

ii. Bounded/Unbounded Error: We require that if the answer is "Yes" then the algorithm is correct with probability strictly greater than 1/2. If the answer is "No" then in the bounded error setting the algorithm is correct with probability at most 1/3, and in the unbounded setting at most 1/2[1].

iii. Halting/Non-Halting: It is senseless to consider algorithms that do not halt with some nonzero probability. Still, we may consider algorithms that *halt almost surely* i.e., for every input the algorithm halts with probability 1. For example, suppose that our TM reads a random bit and stops if it sees a '1' then it does not always halts, but rather halts with probability 1.

We thus have the following four prefixes

$$\mathbf{Pr},\ \mathbf{BP},\ \mathbf{R},\ \mathbf{ZP},$$

respectively referring to unbounded error, bounded two-sided error, (bounded) one-sided error, and (bounded) zero-sided error. For each, we add a sub-script of **H** indicating that the algorithm always halts, and omission of it means that the machine is allowed not to halt with probability zero. These conditions give rise to the following eight types of algorithms that use $O(S)$ space,

$$\begin{array}{ccccccc}
\mathbf{ZP_H SPACE}(S) & \subseteq & \mathbf{R_H SPACE}(S) & \subseteq & \mathbf{BP_H SPACE}(S) & \subseteq & \mathbf{Pr_H SPACE}(S) \\
\rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} & & \rotatebox[origin=c]{-90}{$\subseteq$} \\
\mathbf{ZPSPACE}(S) & \subseteq & \mathbf{RSPACE}(S) & \subseteq & \mathbf{BPSPACE}(S) & \subseteq & \mathbf{PrSPACE}(S)
\end{array}$$

[1]The numbers 1/3, 1/2 could be chosen to be any two quantities that are bounded from each other by a constant.

In terms of the classes defined in Section 2.3

$$\mathbf{BPL} = \mathbf{BP_H SPACE}(\log n),\ \mathbf{RL} = \mathbf{R_H SPACE}(\log n),\ \mathbf{ZPL} = \mathbf{ZP_H SPACE}(\log n).$$

Similarly to the suffix $\mathbf{SPACE}(S)$, we write $\mathbf{TISP}(T, S)$ for algorithms that run $O(T)$ steps *in expectation*, and space $O(S)$. Adding the above prefixes, and the prefix $\mathbf{D}$ for deterministic computation, we obtain five more classes

$$\mathbf{DTISP}(T, S),\ \mathbf{ZPTISP}(T, S),\ \mathbf{RTISP}(T, S),\ \mathbf{BPTISP}(T, S),\ \mathbf{PrTISP}(T, S).$$

Note that in the time-bounded setting there is no need to distinguish between the halting and non-halting models (as the time bound is given). Also, for probabilistic classes we add an additional parameter R for algorithms that use $O(R)$ random bits *in expectation*, e.g.,

$$\mathbf{BPSPACE}(S, R)$$

is the class of bounded two-sided error randomized algorithms that use $O(S)$ space, and $O(R)$ random bits[2]. Again, there is no need to distinguish the halting and non-halting models. Obviously,

$$\mathbf{XSPACE}(S, R) \subseteq \mathbf{XTISP}(R, S)$$

for any $\mathbf{X} \in \{\mathbf{Pr}, \ldots\}$.

Due to a counting argument any algorithm that always halts and run in space S, necessarily runs in at most $2^{O(S)}$ time. Conversely, algorithms that run more than $2^{O(S)}$ time necessarily do not halt for some setting of the randomness tape. Therefore, the value of

$$T = 2^{O(S)}$$

exhibits a *boundary behaviour* where T, S are the time and space complexities respectively. First, the former observation suggests that

$$\mathbf{X_H SPACE}(S) = \mathbf{XTISP}(2^{O(S)}, S)$$

for any $\mathbf{X} \in \{\mathbf{Pr},\ \mathbf{BP},\ \mathbf{R},\ \mathbf{ZP}\}$. Similarly, for $R \geqslant 2^{O(S)}$

$$\mathbf{XSPACE}(S, R) = \mathbf{XTISP}(R, S)$$

as after exhausting all randomness the machine will necessarily stop after $2^{O(S)}$ steps. Despite technically unnecessary, it is perhaps more aesthetic to add the sub-script $\mathbf{H}$ to $\mathbf{BPTISP}(T, S)$, and $\mathbf{BPSPACE}(S, R)$ whenever $T, R < 2^{O(S)}$ indicating that, without the loss of generality, the algorithm always halts.

The following figure is taken from [Sak96] and illustrates the "landscape of randomized space bounded algorithms" as it is known to date. Observe that there is a collapse in the one/zero-sided, non-halting model

$$\mathbf{RSPACE}(S) = \mathbf{coRSPACE}(S) = \mathbf{ZPSPACE}(S) = \mathbf{NSPACE}(S),$$

which is due to Gill [Gil77]. Also, we have left out the unbounded error in the non-halting model since Jung showed in [Jun81] that

$$\mathbf{Pr_H SPACE}(S) = \mathbf{PrSPACE}(S).$$

[2]In Saks's survey [Sak96] he uses the notation $\mathbf{X}_R\mathbf{SPACE}$.

$$\begin{tikzcd}
 & & & \mathbf{DSPACE}(S) \arrow[r, phantom, "\overset{\text{[NZ96]}}{=}"] \arrow[d] & \mathbf{BP_H SPACE}(S, S^{O(1)}) & & \\
 & \mathbf{coR_H SPACE}(S) \arrow[d] \arrow[drr] & & \mathbf{ZP_H SPACE}(S) \arrow[ll] \arrow[rr] & & \mathbf{R_H SPACE}(S) \arrow[d] \arrow[dll] & \\
\mathbf{NSPACE}(S) \arrow[r, phantom, "="] & \mathbf{coRSPACE}(S) & & \mathbf{BP_H SPACE}(S) \arrow[d] \arrow[ddll, "\text{[SZ99, Hoz21]}"'] \arrow[ddrr, "\text{[Nis94]}"] & & \mathbf{RSPACE}(S) \arrow[r, phantom, "\overset{\text{[Gil77]}}{=}"] & \mathbf{NSPACE}(S) \\
 & & & \mathbf{BPSPACE}(S) \arrow[dd] & & & \\
 & \mathbf{DSPACE}(S^{3/2}/\sqrt{\log S}) \arrow[ddrr] & & & & \mathbf{DTISP}(2^{O(S)}, S^2) \arrow[ddll, "\text{[Jun81, BCP83]}"] & \\
 & & & \mathbf{PrSPACE}(S) \arrow[d] & & & \\
 & & & \mathbf{DSPACE}(S^2) & & &
\end{tikzcd}$$

Figure 4.1: The landscape of randomized space bounded algorithms (Figure 3 in [Sak96]). Arrows indicate inclusion.

4.1.2 The Matrix Multiplication Problem

As already indicated in Section 2.4, derandomizing space-bounded algorithms can be reduced to approximating powers of stochastic matrices. Let us now put the problem of matrix multiplication in a broader context, and try to paint an analogous landscape similar to that in Figure 4.1.

The computational complexity of matrix multiplication is arguably one of the most studied problems in the theory of computation, and its importance cannot be overstated. In the time bounded regime, the matrix multiplication problem was granted a special constant, also known as the *matrix multiplication constant*: the smallest constant ω for which any $n \times n$ matrix can be multiplied via

$$O(n^{\omega+o(1)})$$

arithmetic operations. The constant ω frequently appears in the running time complexity of algorithms stressing that a better matrix multiplication algorithm yields improved performance. In the space-bounded regime it is possible to multiply matrices in logarithmic space (See Section 2.2), which is essentially optimal, and so one is mainly interested in matrix powering instead. Similarly to the matrix multiplication constant, the complexity of matrix powering has its own complexity class - **DET**. As the name suggests, **DET** is the class of all problems that reduce[3] to computing the determinant of a given integer matrix, which can be shown to be equivalent to powering integer matrices (and many other linear-algebraic problems such as inversion). In terms of space complexity, it is known that [Ber84, Csa76]

$$\mathbf{DET} \subseteq \mathbf{DSPACE}(S^2),$$

[3]The reduction is required to run in logarithmic space.

which is believed to be tight. For more details on the class **DET** the reader is referred to [Coo85].

The matrix multiplication problem has many natural variants. While many of these often come up in various applications, the universality of matrix multiplication makes them interesting on their own right. We list some of these variants:

i. Assuming that the matrices are of a certain type. For example, many important graph algorithms work with the transition matrix of a graph - a stochastic matrix. Other natural matrix classes include unitary matrices, doubly-stochastic matrices, matrices of bounded norm, M-matrices, Z-matrices etc.

ii. Different notion of approximation: exact computation, or approximation with respect to some notion (e.g., matrix norms).

iii. Powering and Iterated Products: Given n compute the n-th power of a matrix, or more generally, given n matrices $A_1, \dots, A_n$ compute their iterated products. It is also useful to compute a polynomial in a given matrix (e.g., see [CCL+15])

$$p(A) = c_0 I + c_1 A + c_2 A^2 + \cdots + c_d A^d,$$

where $p(x) = \sum_{i=0}^{d} a_i x^i$, or even functions[4], e.g., matrix exponential e^A.

iv. Large/Small Matrices: Compute or approximate the iterated product of n matrices which are $w \times w$ where n, w are not necessarily comparable, i.e., $w \ll n$, or $w \gg n$.

Focusing on space-bounded algorithms, matrix multiplication is not merely an important computational problem, but rather captures computation itself. In high level, the execution of an algorithm can be understood as a *local* operator on the *space of configurations* (See Section 2.4). Here, locality refers to the fact that at any moment a TM can only access one bit of storage, and it can only decide based on that information and its internal memory (which is constant). While in most settings locality is crucial, for low-space algorithms the work-tape is extremely short so locality is usually (if not always) neglected. Assuming this negligence, a low-space algorithm is a successive application of an arbitrary operator on the space of configurations. Such operator can easily be described via a matrix. Deterministic computation corresponds to Boolean stochastic matrices, randomized computation corresponds to stochastic matrices, and even quantum computation (roughly) corresponds to matrices of bounded-norm [FR21].

Going back to the landscape illustrated in Figure 4.1, the problem of approximating the n-th power of a $w \times w$ stochastic matrix is "complete" for the class

$$\mathbf{BPSPACE}(\log w, n),$$

namely randomized algorithms that use $\log w$ space, and n random bits. Consequently, whenever n, w are polynomially related then this problem is "complete" for **BPL**. For the non-halting model of

$$\mathbf{BPSPACE}(\log w)$$

it is tempting to consider the problem computing the infinite power series of a stochastic operator $A \in \mathbb{R}^{w \times w}$

$$I + A + A^2 + A^3 + \cdots,$$

[4]Assume $f\colon \mathbb{R} \to \mathbb{R}$ has a power series $f = \sum_{n=0}^{\infty} a_n x^n$ then we can define $f(A) \overset{\text{def}}{=} \sum_{n=0}^{\infty} a_n A^n$.

as it encodes all possible powers, and hence accounts for all possible halting times. Assuming the above series converges, then by the formula for the sum of geometric series

$$(I - A)^{-1} = I + A + A^2 + A^3 + \cdots .$$

Unfortuantely, the series does not necessarily converge, though we may consider instead what is called the *pseudoinverse* of $I - A$

$$(I - A)^{\dagger} \stackrel{\text{def}}{=} \lim_{\delta \to 0^+} (I - (1 - \delta)A)^{-1}.$$[5]

For classes of type **Pr**, **R**, **ZP** it is possible to consider similar variants. To be exact, the above computational problems are not complete in the traditional sense as they are not a decision problem. The completeness of such problems can be formalized and formulated using the concept of promise problems, though we will not do it here (see [RTV06, Appendix A.2]). Nonetheless, any algorithm for the matrix powering problem yields an upper bound for the corresponding class. E.g., an algorithm of type **X** for approximating the n-th power of a $w \times w$ stochastic operator using $T(n, w)$ time, and $S(n, w)$ space gives

$$\mathbf{BP_H TISP}(n, \log w) \subseteq \mathbf{XTISP}(T(n, w), S(n, w)).$$

Conversely, the aforementioned problems can be solved via a randomized algorithm of the appropriate type that interprets the matrix as a Markov chain (or a graph), and estimates its transition matrix by sampling random walks on it. Again, the reader is referred to Saks's survey for more details ([Sak96, Section 2.4]).

4.2 Our Result

4.2.1 Previous Works

We study the problem of approximating powers of stochastic matrices. Apart from the naive algorithm presented in Section 2.2, there are two non-trivial algorithms for this problem. The first algorithm is based on the celebrated Cayley–Hamilton theorem.

Theorem 4.2.1 ([MP00])**.** *For any $n, w \in \mathbb{N}$ there exists a deterministic algorithm that on input a $w \times w$ matrix A, represented by k bits, outputs A^n using space $O(\log n + \log^2 kw)$.*

For completeness, we give the formal details in Section 4.9.1. It is also worth noting that the algorithm in Theorem 4.2.1 is exact, and works for any matrix.

For stochastic matrices, introducing $\varepsilon > 0$ approximation error, Saks and Zhou devised an algorithm [SZ99] that runs in space

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

The dependence on ε was recently improved by Ahmadinejad, Kelner, Murtagh, Peebles, Sidford, and Vadhan [AKM+20] using the Richardson iteration (see Section 3.5), obtaining

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

[5]This definition is only valid whenever A is stochastic. There is a general way of defining a pseudoinverse but we will not go into that.

space.

Another result that pertains approximating powers of stochastic matrices is due to the aforementioned work of [AKM$^+$20] who showed that the powers of *doubly-stochastic* matrices can be computed in nearly-optimal space complexity.

Theorem 4.2.2 ([AKM$^+$20])**.** *For any $n, w \in \mathbb{N}$ there exists a deterministic algorithm that given a $w \times w$ doubly-stochastic matrix A, represented by k bits, outputs A^n using*

$$O\left(\log(nkw)\log\log\frac{nkw}{\varepsilon}\right).$$

space.

A doubly-stochastic matrix, is a matrix whose entries are non-negative, and all its rows and columns sum to 1. Hence, doubly-stochastic matrices form a sub-class of stochastic matrices. Furthermore, for matrix powering their algorithm essentially applies to any stochastic matrix with known *stationary distribution*[6], e.g., regular graphs (a.k.a. Eulerian graphs). Their result is an accumulation of a long line of works [RVW02, Rei08, RV05, RTV06, CKP$^+$16, CKP$^+$17, MRSV17, MRSV19].

4.2.2 Our Result

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

Theorem 4.2.3 ([CDSTS22])**.** *For any $w, n \in \mathbb{N}$, and $\varepsilon > 0$, there exists a deterministic algorithm that given stochastic matrix $A \in \mathbb{R}^{w\times w}$ approximates the power A^n to within accuracy $\varepsilon = n^{-\log n}$ 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.

The algorithm of Theorem 4.2.3 outperforms previous results whenever

$$w \ll n.$$

For concreteness, let us take $w = 2^{\log^\alpha n}$ for some constant $\alpha \in (0,1)$. Assuming $w \leqslant n$, the Saks–Zhou algorithm requires $O(\log^{3/2} n)$ space, and the algorithm that is given by theorem 4.2.1 requires $O(\log^{1+2\alpha} n)$ space. We can then conclude a slight improvement to the landscape depicted in Figure 4.1 proving

$$\mathbf{BPSPACE}(S,R) \subseteq \mathbf{DSPACE}(\log R + S\sqrt{\log R}),$$

which improves upon the Saks–Zhou result in the regime of $R > \log S$. Recall that in this regime, the algorithm necessarily does not halt, and also this is the same as

$$\mathbf{BPTISP}(T,S) \subseteq \mathbf{DSPACE}(\log T + S\sqrt{\log T}).$$

[6]Suppose that G is a strongly connected graph, with transition matrix A then by the Perron–Frobenius theorem there exists a unique distribution π such that $A\pi = \pi$. The distribution π is called the *stationary distribution* of G.

4.3 Overview

We explain why the Saks–Zhou algorithm fails to provide better parameters in the regime $w \ll n$, and proceed to describe how to adjust it in order to get the improved parameters of Lemma 5.4.1.

4.3.1 The Saks–Zhou Framework

The Saks–Zhou algorithm consists of two ingredients:

1. The celebrated Nisan generator (see Section 2.7.1), which is used as a randomized matrix exponentiation algorithm.
2. *Canonization* using the *shift and truncate* technique (see Section 4.5). 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).

Roughly speaking, the Saks–Zhou algorithm works as follows. 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) Approximate the $2^{\sqrt{\log n}}$-th power of $\widetilde{M}_{i-1}$ within accuracy ρ_1 using the Nisan generator.
   (b) Shift that approximation by a random shift of magnitude $\zeta \cdot \rho_2$, where ζ is chosen uniformly from $\{0, 1, \dots, L\}$, and truncate it to a precision of ρ_2.
   (c) Set $\widetilde{M}_i$ to be the result of that shift and truncation.
3. Output $\widetilde{M}_i$ for $i = \sqrt{\log n}$.

The crucial point in the above procedure is that the canonicalization step (to be discussed later) enables the *reuse* of the randomness needed for the different applications of the Nisan generator. Specifically:

- Applying the Nisan generator to approximate $A^{2^{\sqrt{\log n}}}$, for an $w \times w$ matrix A, requires seed length of
$$O\left(\log \frac{n}{\rho_1} \cdot \sqrt{\log n}\right)$$
bits that are **fixed and reused** throughout the different applications of the Nisan generator.
- Drawing the shifts requires
$$\log L \cdot \sqrt{\log n}$$
bits.
- Each of the $\sqrt{\log n}$ steps requires $O(\log \frac{n}{\rho_1})$ space.

For simplicity, let us first consider the case $w = n$. It turns out that ρ_1, ρ_2, and L all have to be polynomial in n namely,

$$\rho_1 = n^{-\Theta(1)}, \text{ and } \rho_2 = n^{-\Theta(1)}, \text{ and } L = n^{\Theta(1)}.$$

The reason why ρ_1 and ρ_2 have to be polynomially small is because errors accumulate additively, and if we raise to a power of n, the final error contains a term of order

$$n(\rho_1 + \rho_2).$$

Also, L has to be polynomially large because for each of the w^2 entries of A, the shift has probability at least $1/L$ to be "bad", and therefore the final error term contains a term of order

$$\frac{w^2}{L}.$$

Altogether, the space complexity becomes $O(\log^{3/2} n)$.

Let us now check what changes when we separate the dimension parameter w from the exponent parameter n, and when $w \ll n$. Suppose our input is a $w \times w$ stochastic matrix A, and we want to approximate A^n. For simplicity, we employ the same approach as before: i.e., we have $\sqrt{\log n}$ steps, each raising to a power of $2^{\sqrt{\log n}}$. Then:

- As before, the parameters ρ_1, ρ_2 have to be $n^{-\Theta(1)}$, because the errors accumulate additively in the exponent parameter n.
- However, L may be smaller now, as there are only w^2 entries in the matrix A, and we only need to take a union bound over $w^2\sqrt{\log n}$ events (w^2 events for each of the $\sqrt{\log n}$ steps). Indeed, our algorithm invests roughly $\log w$ random bits for choosing the shifts[7].

Doing the calculation of the overall space complexity we see that we have gained nothing since the space complexity of a single application of the Nisan generator is still

$$O(\sqrt{\log n} + \log w + \log \frac{1}{\rho_1}) = O(\log n).$$

In fact, any approximation of precision ρ_1 that comes from using a PRG must have space complexity $\Omega(\log \frac{1}{\rho_1})$. The crux of the problem lies in the fact that we have to work with accuracy $n^{-\Omega(1)}$, and then it seems inevitable that each step of the $\sqrt{\log n}$ steps should have space complexity $O(\log n)$, yielding the same space complexity of $O(\log^{3/2} n)$ as before.

In order to avoid the aforementioned loss of parameters in the case $w \ll n$ we employ the following approximation scheme: Throughout the computation our matrices will be kept with $n^{-\Theta(1)}$ accuracy. However, we purposely *decrease* the precision of the input matrix to the Nisan generator by truncating its entries to a precision of $w^{-\Omega(1)}$. 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 $n^{-\Omega(1)}$, we invoke the *Richardson iteration* (See Section 3.5). It is crucial to note that although we decrease the precision, 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 algorithm (see also Figure 4.2). The precise description is given in Section 4.6. 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 $\widetilde{M_0} = A$.

[7]Note, however, that each shift has magnitude at most $L \cdot \rho_1 = n^{-\Theta(1)}$.

2. For $i = 1, \ldots, \sqrt{\log n}$,
   (a) Truncate $\widetilde{M}_{i-1}$ to a precision of $w^{-\Omega(1)}$ and denote this by $\lfloor \widetilde{M}_{i-1} \rfloor$.
   (b) Set the Nisan generator to work with accuracy $w^{-\Omega(1)}$ 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 R_i by a random shift of magnitude $n^{-\Omega(1)}$, and truncate it to a precision of $n^{-\Omega(1)}$. We set $\widetilde{M}_i$ to be the result of that shift and truncation.
3. Output $\widetilde{M}_i$ for $i = \sqrt{\log n}$.

Figure 4.2 illustrates the alternating nature of the algorithm, zig-zagging between a mild approximation of $w^{-\Omega(1)}$ and a high precision approximation of $n^{-\Omega(1)}$. 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

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

space.

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

Figure 4.2: Our Improved SZ Algorithm. "S & T" refers to "shift and truncate".

Averaging over the auxiliary randomness, as done in [SZ99], would yield a space-efficient deterministic algorithm, albeit with accuracy of $w^{-\Omega(1)}$. It is thus tempting to try and apply an additional layer of the Richardson iteration in order to improve the accuracy to an arbitrary $\varepsilon > 0$ (similar to [AKM+20] for the standard Saks–Zhou algorithm when $w = n$). However, to apply the Richardson iteration, the initial accuracy needs to be at least $n^{-\Omega(1)} \ll w^{-\Omega(1)}$. 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.

4.4 Preliminaries

4.4.1 Matrix Sequences

We shall denote a sequence of matrices by (M) namely

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

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

$$(M)_B \overset{\text{def}}{=} ((M)_{b_1}, (M)_{b_2}, \dots).$$

We extend operations to sequences element-wise, in the natural way. E.g., if T is some operator on matrices then $T((M))$ is the sequence defined by $T((M))_j \overset{\text{def}}{=} T((M)_j)$. Similarly, we define the following shorthand

$$\prod_B (M) \overset{\text{def}}{=} \prod_j (M)_{b_j}.$$

4.4.2 The Richardson Iteration

Recall the Richardson iteration from Section 3.5. Previously, we have used it in Chapter 3 to derive a WPRG against the model ROBPs, though a more straightforward application of it is an algorithm that improves the accuracy of matrix iterated products [AKM+20, PV21, CDR+21]. For completeness, we provide the short proof in section 4.9.2.

Lemma 4.4.1. *There exists an algorithm* R *that gets as input a sequence of sub-stochastic matrices* $(A) = (A_1, \dots, A_n)$ *of dimension* $w \times w$*, an integer* $k \in \mathbb{N}$*, and a sequence sub-stochastic matrices* (B) *satisfying:*

- *If for all* $1 \leqslant i \leqslant j < n$
$$\|A_i \cdots A_j - (B)_{i,j}\|_\infty \leqslant \frac{1}{4(n+1)},$$
then
$$\left\|\mathsf{R}\big((B_{i,j})_{i,j=1}^n, (A_i)_{i=1}^n, k\big) - A_1 \cdots A_n\right\|_\infty \leqslant (n+1) \cdot 2^{-k}.$$
- R *runs in*
$$O\big(\log^2 k + \log k \cdot \log(nT)\big)$$
space, where $T = \max\{|A_i|, |(B)_{i,j}|\}$ *is the maximum bit-complexity of the given matrices.*

In the above lemma, whenever $A_1 = A_2 = \cdots = A_n$ then it suffices to get as input matrices $(B) = (B_1, \dots, B_n)$ satisfying
$$\left\|A^i - B_i\right\|_\infty \leqslant \frac{1}{4(n+1)}.$$
In this case, we shall invoke the algorithm using the syntax

$$\mathsf{R}((B), A, k),$$

where $A = A_1 = A_2 = \cdots = A_n$.

4.5 Revisiting the Saks–Zhou Framework

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.

4.5.1 Canonicalization of ROBPs

An important step in [SZ99] is to transform a given stochastic matrix (or a sub-stochastic one) into an ROBP, 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 ROBP of width $w+1$, over alphabet $\Sigma = \{0,1\}^s$ constructed as follows. Given $i \in [w]$ and $\sigma \in \Sigma$, $B(i,\sigma) = j$ where j is the smallest integer satisfying $\sum_{k \leqslant j} M[i,k] \geq \sigma \cdot 2^{-s}$ if such exists, and $w+1$ otherwise. Moreover, we set $B(w+1,\sigma) = w+1$ for all $\sigma \in \Sigma$. The following claim then follows easily.

Claim 4.5.1. *For a sub-stochastic matrix M, it holds that $\mathsf{M}(\mathsf{C}(M))_{[1,w]} = M$, 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 an ROBP 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 an ROBP 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 $\sum_{k \leqslant j} M[i,k] \geq \sigma \cdot 2^{-t}$ if such exists, and $w+1$ otherwise. We also set $B(w+1,\sigma) = w+1$ for all $\sigma \in \Sigma$ as before.

Claim 4.5.2. *Let A be a $w \times w$ sub-stochastic matrix A in which every entry is represented using at most s bits, and let $t \in \mathbb{N}$ where $t \leqslant s$. Then,*

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

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

An Extended Nisan Algorithm. Recall the Nisan generator presented in Section 2.7.1. For simplicity, let us only consider an ROBP of width w over alphabet Σ 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 4.5.3 ([Nis92])**.** *There exists an algorithm N that gets as input an ROBP $B \in ([w] \times \Sigma \to [w])^n$ with a transition matrix $A = \mathsf{M}(B)$, an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and a seed $h \in \{0,1\}^{d_\mathsf{N}}$ where $d_\mathsf{N} = O\Big(\log n \cdot \log \frac{nw|\Sigma|}{\varepsilon\delta}\Big)$. The algorithm runs in space $O\Big(\log \frac{nw|\Sigma|}{\varepsilon\delta}\Big)$ and outputs a sequence*

$$\mathsf{N}(B,h) = (M_h),$$

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

$$\left\| (M_h)_i - A^i \right\|_\infty \leqslant \varepsilon.$$

We will often want to feed Nisan's algorithm with stochastic (or even sub-stochastic) matrices, rather than ROBPs. The following theorem extends upon theorem 4.5.3 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 an ROBP as input and for the one that gets a matrix as input.

Lemma 4.5.4. *There exists an algorithm 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 $t \leqslant s$.*
3. *A seed $h \in \{0,1\}^{d_{\mathsf{N}}}$ for $d_{\mathsf{N}} = O\big(\log n \cdot \big(t + \log \frac{nw}{\varepsilon\delta}\big)\big)$.*

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

$$\mathsf{N}(A, h) = (M_h),$$

each $(M_h)_i \in \mathbb{R}^{w \times w}$, and satisfies the following

$$\Pr_{h \in \{0,1\}^{d_{\mathsf{N}}}} \left[\forall i \ \left\|(M_h)_i - A^i\right\|_\infty \geqslant \varepsilon + nw \cdot 2^{-t}\right] \leqslant \delta.$$

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

Proof. We compute $B = \mathsf{C}_t(A)$ and apply $\mathsf{N}(B, h, n)$, which outputs $(M_h)_1, \ldots, (M_h)_n$. We then consider only the first w rows and columns of each matrix. By Theorem 4.5.3, with probability at least $1 - \delta$ over $h \in \{0,1\}^{d_{\mathsf{N}}}$, we are guaranteed that

$$\left\|(M_h)_i - \mathsf{M}(B)^i\right\|_\infty \leqslant \varepsilon$$

for all $i \in [n]$. By claim 4.5.2, $\|\mathsf{M}(B) - A\|_\infty \leqslant w \cdot 2^{-t}$, and thus, due to claim 2.4.5,

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

The space requirements and the bound for d_{N} readily follows from claim 4.5.2 and theorem 4.5.3. Note that when $t = s$, the canonicalization is lossless. $\square$

4.5.2 Shift and Truncate

Definition 4.5.5 (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,*

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

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

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

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

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

$$\exists a \in \mathbb{N}, \quad a2^{-t} \in \left[z - \zeta 2^{-2t}, y - \zeta 2^{-2t}\right). \tag{4.1}$$

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

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

is disjoint and contained in the interval I which is of length at most $|y - z| + (2^\ell - 1)2^{-2t} \leqslant 2^{-t}$. Hence, there is at most one point in I which is an integer multiple of 2^{-t}, meaning that there is at most one ζ satisfying Equation (4.1). ☐

The preceding lemma is an important ingredient in [SZ99], that enables one to eliminate dependencies between consecutive applications of Nisan's algorithm. Think of z as an approximation to some y obtained by a randomized algorithm that typically returns a good approximation $z \approx y$. Note that while z, y might be extremely close, their truncation may differ if they are on the *boundary* values of the truncation operator. The idea behind Lemma 4.5.6 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 4.5.6 to matrices, a simple union-bound gives us the following corollary.

Corollary 4.5.7. *Let $M, M' \in \mathbb{R}^{w \times w}$ be such that $\|M - M'\|_{\max} \leqslant 2^{-2t}$. Then, for all $\ell < t$, we have that*

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

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

4.5.3 Revisiting 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 $O(r^{3/2})$.

Without the loss of generality we may assume that the input matrix A is given to us using t digits of precision.

```
Algorithm: Saks–Zhou (SZ)
  Input: Stochastic Matrix $A \in \mathbb{R}^{w\times w}$ represented to $t$ bits of accuracy.
  Output: Stochastic Matrix $M \in \mathbb{R}^{w\times w}$ that approximates $A^{2^r}$.
  Parameters: Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ be the desired
  confidence. Set $t = \log \frac{2nw^2 r_2}{\varepsilon\delta}$, $\ell = t/2$. Instantiate $\mathsf{N}$ to approximate powers of order
  $2^{r_1}$ with accuracy $\varepsilon_{\mathsf{N}} = 2^{-2t}$, confidence $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, and canonicalization parameter $t$.
  Also, set $r = \log n = r_1 r_2$.

  1. Draw $h \sim \{0,1\}^{d_{\mathsf{N}}}$ uniformly.
  2. Set $\widetilde{M}_0 = A$.
  3. For $i = 1, \dots, r_2$,
      (a) Draw $\zeta_i \sim \{0, \dots, 2^\ell - 1\}$ uniformly.
      (b) Set $\widetilde{M}_i = \left\lfloor \mathsf{N}\left(\widetilde{M}_{i-1}, h\right) - \zeta_i 2^{-2t} J_w \right\rfloor_t$.
  4. Output $\widetilde{M}_{r_2}$.
```

Theorem 4.5.8 ([SZ99])**.** *For any $w \times w$ stochastic matrix A, and integers r_1, r_2 such that $r_1 r_2 = r = \log n$*

$$\Pr_{h,\zeta}[\|\mathsf{SZ}(A,h,\zeta) - A^n\|_\infty \geqslant \varepsilon] \leqslant \delta.$$

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

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

$$M_i(\zeta) = \lfloor M_{i-1}(\zeta)^{2^{r_1}} - \zeta_i 2^{-2t} J_w \rfloor_t. \tag{4.2}$$

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

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

$$\mathrm{GOOD}_{i,\zeta} = \{h \in \{0,1\}^{d_{\mathsf{N}}} : \big\|M_i^{2^{r_1}} - \mathsf{N}(M_i,h)\big\|_\infty \leqslant \varepsilon_{\mathsf{N}}\}. \tag{4.3}$$

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 lemma 4.5.4, we get that for any $i \in [r_2]$ and $\zeta_1, \dots, \zeta_i$,

$$\Pr_{h\in\{0,1\}^{d_{\mathsf{N}}}}[h \in \mathrm{GOOD}_{i,\zeta}] \geq 1 - \delta_{\mathsf{N}}. \tag{4.4}$$

Claim 4.5.9. *It holds that*

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

Proof. We prove by induction on i that

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

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

$$E = \{(h, \zeta) : \exists j < i \text{ such that } M_j \neq \widetilde{M_j} \text{ or } h \notin \mathrm{GOOD}_{i-1,\zeta}\}.^{8}$$

Next, we write

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

By the induction's hypothesis the first term is at most $(\delta_{\mathsf{N}} + w^2 2^{-\ell})(i-1)$, and by Equation (4.4) the second term is at most δ_{N}, so it suffices to show that

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

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

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

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

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

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

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

as well. By corollary 4.5.7, with probability at least $1 - w^2 2^{-\ell}$ over ζ_i,

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

which simply amounts to $M_i = \widetilde{M_i}$, as desired (see eq. (4.2) for the definition of M_i). This completes the inductive step. □

Next, we handle the accuracy guarantee.

Claim 4.5.10. *For all $i \in \{0, 1, \ldots, r_2\}$ and all ζ it holds that*

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

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

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

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

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

and so by Claim 2.4.4, $\left\|M_i - M_{i-1}^{2^{r_1}}\right\|_\infty \leqslant 2^{-t+1}w$. Write

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

By the induction's hypothesis and Claim 2.4.5, 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

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

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

claim 4.5.9 tells us that with probability at least $1-\delta$, $\widetilde{M}_{r_2} = M_{r_2}$. By claim 4.5.10 above, $\left\|\widetilde{M}_{r_2} - A^n\right\| \leqslant \varepsilon$, so the correctness follows.

For the space complexity, by lemma 4.5.4, 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 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.

4.6 Algorithm Outline, Correctness, and Complexity

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 4.3. To this end, we devise a randomized algorithm which is derandomized in section 4.8, state its correctness, and space complexity.

```
Algorithm: Improved Saks–Zhou ($\mathsf{SZ}_{\mathsf{Imp}}$)
  Input: Stochastic Matrix $A \in \mathbb{R}^{w \times w}$ represented using $t_2$ bits of accuracy.
  Output: Stochastic Matrix $M \in \mathbb{R}^{w \times w}$ that approximates $A^{2^r}$.
  Parameters: Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ be the desired
  confidence. Set $t_1 = 4r_1 + \log w$, $t_2 = \log \frac{16nw^2 r_2}{\varepsilon\delta}$, $\ell = \log \frac{2w^2 r_2}{\delta}$. Instantiate $\mathsf{N}$ to
  approximate powers of order $2^{r_1}$ with accuracy $\varepsilon_{\mathsf{N}} = 2^{-2r_1}$, confidence $\delta_{\mathsf{N}} = \frac{\delta}{2r_2}$, and
  canonicalization parameter $t_1$. Also, set $r = \log n = r_1 r_2$.

  1. Draw $h \sim \{0,1\}^{d_{\mathsf{N}}}$ uniformly.
  2. Set $\widetilde{M}_0 = A$.
  3. For $i = 1, \dots, r_2$,
     (a) Draw $\zeta_i \sim \{0, 1, \dots, 2^\ell - 1\}$ uniformly.
     (b) Compute $\widetilde{M}_i = \left\lfloor \mathsf{R}\left(\mathsf{N}\left(\widetilde{M}_{i-1}, h\right), \widetilde{M}_{i-1}, 3t_2\right) - \zeta_i 2^{-2t_2} J_w \right\rfloor_{t_2}$.
  4. Output $\widetilde{M}_{r_2}$.
```

Remark 9. *The Richardson iteration may output a matrix which is not sub-stochastic, but as we shall see from the analysis, if this happens the algorithm can simply 'quit'.*

The parameters are chosen to satisfy the following constraints:

$$\ell < t_2, \tag{4.5}$$

$$t_1 < t_2, \tag{4.6}$$

$$\varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1} \leqslant \frac{1}{4(2^{r_1}+1)}, \tag{4.7}$$

$$2^{-t_2} \leqslant \frac{1}{2^{r_1}+1}. \tag{4.8}$$

Equations (4.5) and (4.6) are so that we can apply Corollary 4.5.7, and Equations (4.7) and (4.8) are so that we can apply the Richardson iteration (Lemma 4.4.1).

Lemma 4.6.1. *For any $w \times w$ stochastic matrix A, and integers r_1, r_2 such that $r_1 r_2 = r = \log n$*

$$\Pr_{h,\zeta}\left[\|A^n - \mathsf{SZ}_{\mathsf{Imp}}(A, h, \zeta)\|_\infty \geqslant \varepsilon\right] \leqslant \delta.$$

Before delving into the analysis, let us briefly discuss our parameters. We start with the truncation parameters t_1, t_2. The parameter t_2 determines the accuracy of the algorithm, and is governed by n. The second truncation occurs implicitly within the canonicalization step where 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 "shift" parameter ℓ is 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 ℓ does not depend on n and the accuracy parameter ε. In contrast, in [SZ99], $\ell = \Omega(t)$ (see section 4.5).

Next, we determine our algorithm's space complexity.

Lemma 4.6.2. *The algorithm* $\mathsf{SZ}_{\mathsf{Imp}}(A, h, \zeta)$ *can be implemented in*

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

space.

Proof. Consider the function $f(\widetilde{M_i}) = \widetilde{M_{i+1}}$ describing one iteration of item 3. 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:

- The Nisan generator N: By Lemma 4.5.4, 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 4.4.1, 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 2.2.3 we can sum the above and multiply by r_2, obtaining our desired overall space complexity. □

4.7 Proof of Correctness

Throughout the analysis we shall assume that Equations (4.5), (4.6), (4.7), (4.8) hold.

Proof of Lemma 4.6.1. We let M_i be the "true" random rounding, similar to the analysis in section 4.5. That is, $M_0 = A$, and for each $i \in [r_2]$,

$$M_i(\zeta) = \left\lfloor M_{i-1}(\zeta)^{2^{r_1}} - \zeta_i 2^{-2t_2} J_w \right\rfloor_{t_2}. \tag{4.9}$$

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

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

$$\mathrm{GOOD}_{i,\zeta} = \{h \in \{0,1\}^{d_{\mathsf{N}}} : \forall j \leqslant 2^{r_1}, \left\| M_i^j - \mathsf{N}(M_{i-1}, h)_j \right\|_\infty \leqslant \varepsilon_{\mathsf{N}} + w \cdot 2^{r_1 - t_1}\}, \tag{4.10}$$

where $\left(M_i^{(1)}, \ldots, M_i^{(2^{r_1})}\right) = \mathsf{N}(M_i, h)$. (Note that here we feed N with M_i and not $\widetilde{M_i}$, similar to what we did in section 4.5.) By lemma 4.5.4, we get that for any $i \in [r_2]$ and $\zeta = (\zeta_1, \ldots, \zeta_i)$,

$$\Pr_{h \in \{0,1\}^{d_{\mathsf{N}}}}[h \in \mathrm{GOOD}_{i,\zeta}] \geq 1 - \delta_{\mathsf{N}}. \tag{4.11}$$

Claim 4.7.1. *It holds that*

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

Proof. The proof is similar to the proof of claim 4.5.9. We prove by induction on i that

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

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

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

and again we have

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

By the induction's hypothesis the first term is at most $(\delta_{\mathsf{N}} + w^2 2^{-\ell}) \cdot (i-1)$, and by Equation (4.11) the second term is at most δ_{N}. Thus, it suffices to show that

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

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

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

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

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

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

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

Using Lemma 4.4.1 and the guarantee of Equations (4.7) and (4.8), we get

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

Thus, by Corollary 4.5.7, with probability at least $1 - w^2 2^{-\ell}$ over ζ_i,

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

and recalling the definition of M_i from Equation (4.9) we see that it simply means that $M_i = \widetilde{M_i}$. This completes the inductive step. □

For the accuracy guarantee, we have

Claim 4.7.2. *For all $i \in \{0, 1, \ldots, r_2\}$ and all ζ it holds that*

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

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

The proof is identical to the proof of claim 4.5.10, so we omit it. To conclude, note that by claim 4.7.1 we have that with probability at least $1 - \delta$, $\widetilde{M}_{r_2} = M_{r_2}$, and by claim 4.7.2 we establish the accuracy guarantee $\left\| \widetilde{M}_{r_2} - A^n \right\| \leq \varepsilon$. The space requirement was established in lemma 4.6.2. □

We note that by averaging over all seeds (namely, h and the ζ-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.

4.8 A High-Accuracy Deterministic Approximation

In this section we prove our main result, Lemma 4.8.1, 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 Lemma 4.6.1 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. Thus, to gain any improvement we cannot afford to take $\delta \approx \varepsilon$ as would be the case if we "mixed" the δ 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, ζ)-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 $\{A_1, \ldots, A_m\}$, we denote by $\text{median}_{i \in [m]} A_i$ the matrix M for which $M[a, b]$ is the median of $A_i[a, b]$ over all $i \in [m]$.

Algorithm: Improved Saks–Zhou ($\mathsf{SZ}^{+}_{\mathsf{Imp}}$)

Input: Stochastic Matrix $A \in \mathbb{R}^{w \times w}$.

Output: Stochastic Matrix $M \in \mathbb{R}^{w \times w}$ that approximates A^n.

Parameters: Let $\varepsilon > 0$ be a desired accuracy parameter, and $\delta > 0$ be the desired confidence. Instantiate $\mathsf{SZ}_{\mathsf{Imp}}$ with $\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} = \frac{1}{8w(n+1)\log n}$, $\delta_{\mathsf{SZ}_{\mathsf{Imp}}} = 1/4$. The parameters r_1, r_2 are instantiated therein.

1. For $i = 1, \ldots, \log n$, compute
$$\widetilde{M}_{2^i} = \underset{h,\zeta}{\mathrm{median}}\ \mathsf{SZ}_{\mathsf{Imp}}\left(\lfloor A \rfloor_{t_2}, h, \zeta\right),$$
with $r_1 = r_2 = \sqrt{i}$ for approximating the 2^i-th power.

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

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

Remark 10. *For simplicity we assume $\sqrt{i}$ is an integer (although it is clearly not). A more precise is to consider $\lfloor \sqrt{i} \rfloor$ instead and approximate the $2^{\lfloor \sqrt{i} \rfloor^2}$-th power using the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm. As $i - \lfloor \sqrt{i} \rfloor^2 \leq 2\sqrt{i} + 1$, computing the "missing" $2^{O(\sqrt{i})}$ power can be done via the naive matrix multiplication algorithm (Claim 2.2.5) without effecting the overall space complexity.*

For the above choice of parameters, we get that the truncation parameter t_2 from Section 4.6 satisfies
$$t_2 = \log \frac{16 w^2 r_2 n}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} \delta_{\mathsf{SZ}_{\mathsf{Imp}}}} = O\left(\log nw\right).$$
Also, note that d_{N} in $\mathsf{SZ}_{\mathsf{Imp}}$ satisfies
$$d_{\mathsf{N}} = O\left(r_1^2 + r_1 \cdot \log \frac{r_2 w}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right) = O\left(\log n + \sqrt{\log n} \cdot \log w\right),$$
and the number of bits needed to represent $\zeta_1, \ldots, \zeta_{r_2}$ is given by
$$|\zeta| = O\left(r_2 \cdot \log \frac{r_2 w}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right) = O\left(\sqrt{\log n} \cdot \log\log n + \sqrt{\log n} \cdot \log w\right).$$
The proof of Theorem 4.2.3 is now a corollary of the following statement.

Lemma 4.8.1. *Given a $w \times w$ stochastic matrix A, the algorithm $\mathsf{SZ}^{+}_{\mathsf{Imp}}$ above satisfies*
$$\left\| A^n - \widetilde{M} \right\|_\infty \leq \varepsilon,$$
and runs in space
$$O\left(\left(\log n + \sqrt{\log n} \cdot \log w\right) \cdot \log\log nw + \left(\log\log \frac{1}{\varepsilon}\right)^2 + \log\log \frac{1}{\varepsilon} \cdot \log(nw)\right).$$

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 Lemma 4.6.1, the $\mathsf{SZ}_{\mathsf{Imp}}$ algorithm with the prescribed parameters takes

$$O\left(\left(\log n + r_2\log w\right)\log\log\frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}\delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2\log\frac{1}{\delta_{\mathsf{SZ}_{\mathsf{Imp}}}} + r_2\left(\log\log\frac{nw}{\varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}\delta_{\mathsf{SZ}_{\mathsf{Imp}}}}\right)^2\right)$$

  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 $a_1,\ldots,a_m$ 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 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 4.4.1, 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 2.2.2).

We now proceed with the correctness. By Lemma 4.6.1, for at least $\frac{3}{4}$ of the (h,ζ)-s, we have

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

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

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

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

$$\left| \left(\operatorname*{median}_{h,\zeta} \ \mathsf{SZ}_{\mathsf{Imp}} \left(\lfloor A \rceil_{t_2}, h, \zeta \right) \right) [a, b] - \lfloor A \rceil_{t_2}^{2^i} [a, b] \right| \leqslant \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}}.$$

This is true for all indices (a, b) and all $i \in [\log n]$ and so by Claim 2.4.4, for all $i \in [\log n]$,

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

By Claim 2.4.5, $\left\| A^j - \lfloor A \rceil_{t_2}^j \right\|_\infty \leqslant jw2^{-t_2}$, and applying Claim 2.4.5 again for multiplication of $\log n$ matrices, we get that for every $j \leq n$,

$$\left\| \widetilde{M}_j - A^j \right\|_\infty \leqslant \left\| \widetilde{M}_j - \lfloor A \rceil_{t_2}^j \right\|_\infty + \left\| \lfloor A \rceil_{t_2}^j - A^j \right\|_\infty \leqslant \varepsilon_{\mathsf{SZ}_{\mathsf{Imp}}} w \log n + nw2^{-t_2} \leqslant \frac{1}{4(n+1)}.$$

Using Lemma 4.4.1, we obtain

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

which completes the proof. □

4.9 Appendix

4.9.1 Spectral Algorithm for Matrix Powering

In this section we prove Theorem 4.2.1. 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 4.9.1 ([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 4.9.2 ([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 4.9.3. *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 4.9.1, Item 1 can be done in $O(\log w \cdot \log |A|)$ space. Item 2, following Theorem 4.9.2, 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 2.2.6. 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

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

□

4.9.2 Proof of Lemma 4.4.1

In this section, for completness, we prove Lemma 4.4.1.

Proof of Lemma 4.4.1. 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 Section 3.5.

The space complexity of the algorithm follows by Claim 2.2.6. 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 $\left\|A^{-1}[i,j] - B[i,j]\right\|_\infty \leqslant \frac{1}{4(n+1)}$ for every $0 \leqslant i, j \leqslant n$, and so

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

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

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

The correctness now follows by Lemma 4.4.1.

□

Chapter 5

Approximating Products of Stochastic Matrices in Small Space

5.1 Background

In light of the powering algorithm of Chapter 4 it is natural to ask: what is the space complexity required to approximate the iterated product

$$A_1 \cdot A_2 \cdots A_n$$

of $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$ stochastic? There is a standard reduction from matrix iterated product to matrix powering: Let $A_1, \ldots, A_n \in \mathbb{R}^{w \times w}$ then the $[1, n+1]$-th entry of

$$\begin{pmatrix} 0 & A_1 & \ldots & 0 & 0 \\ 0 & 0 & A_2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ \vdots & & & \ddots & A_n \\ 0 & 0 & \ldots & \ldots & 0 \end{pmatrix}^n$$

is exactly the iterated product $A_1 \cdots A_n$. The downside is that the above matrix is of dimension nw (rather than w). Combining the reduction with the Cayley–Hamilton based algorithm (Theorem 4.2.1) yields space complexity of $O(\log^2 nw)$, worse than the naive algorithm (Claim 2.2.7). However, combining it with the Saks–Zhou algorithm gives the same non-trivial space complexity of

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

The algorithm of [AKM+20] for computing powers of doubly-stochastic matrices does extend to iterated products using the above reduction, and so for iterated products of doubly-stochastic matrices we get the nearly-optimal space complexity of

$$O\Big(\log nw \log\log \frac{nw}{\varepsilon}\Big).$$

As previously mentioned, their **powering** algorithm extends to any stochastic matrix with known stationary distribution, though it is unclear thus whether it extends to iterated products, namely: given n matrices $A_1, \ldots, A_n$ along with their stationary distribution $\pi_1, \ldots, \pi_n$ approximate the product

$$A_1 \cdots A_n.$$

While technically there is not much sense in considering such products it does indicate a gap in our understanding. One might argue that an optimal algorithm for approximating powers of **general** stochastic matrices yields an optimal algorithm for the iterated product problem, suggesting that we should focus on matrix powering. On the other hand, it might be unreasonable to expect any progress for the matrix powering problem that does not go through the iterated product problem.

5.2 Our Result

We extend the algorithm presented in Chapter 4 to iterated products.

Theorem 5.2.1 ([CDSTS22])**.** *For any $w, n \in \mathbb{N}$, and $\varepsilon > 0$, there exists a deterministic algorithm that given n stochastic matrices $A_1, \dots, A_n \in \mathbb{R}^{w \times w}$ approximates the iterated product*

$$A_1 A_2 \cdots A_n$$

to within accuracy $\varepsilon = n^{-\log n}$ in space

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

where the $\widetilde{O}$ notation hides doubly-logarithmic factors in n and w. More precisely, our algorithm requires

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

space.

For concreteness let us focus on the regime $w = 2^{\sqrt{\log n}}$. As explained above, the algorithm of Theorem 4.2.1 does not extend to iterated products, and the Saks–Zhou algorithm requires

$$O(\log^{3/2} n)$$

space, which exactly matches the space complexity of the naive algorithm Claim 2.2.8. In fact, for the problem of computing the iterated product in the case $w = 2^{\sqrt{\log n}}$ nothing better was known apart from the naive algorithm, whereas our algorithm achieves nearly-optimal space complexity of

$$O(\log n \log\log n).$$

5.3 Overview

Let us try to naively extended the powering algorithm suggested in Chapter 4 to iterated products as follows:

1. Use the 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 naive attempt:

1. The Shifts: Previously, we used a fresh shift in every iteration. In the case of iterated product, there are many input matrices. We cannot afford to perturb each of those via a distinct independent shift. Can we use the same shift on all the different matrices? If not, what else can we do?

2. The Confidence Parameter: In the matrix powering algorithm the Nisan generator has to work against any power

$$A, A^{2^{\sqrt{\log n}}}, A^{2^{2\sqrt{\log n}}}, A^{2^{3\sqrt{\log n}}}, \ldots, A^n.$$

   As there are only $\sqrt{\log n}$ such powers, we could choose a large confidence parameter δ_{N} (see, e.g., Section 2.7.1 or Lemma 4.5.4) and still be sure that with a good probability over h, our choice works well for all the $\sqrt{\log(n)}$ matrices above.

   In contrast, for the iterated matrix product algorithm, we need to fix a single h that works well against any sub-product, and there are n such products. Therefore, the confidence deteriorates to $n \cdot \delta_{\mathsf{N}}$ which forces us to take δ_{N} smaller than $\frac{1}{n}$. However, in this parameter setting the Nisan generator has seed length $\Omega(\log^{3/2} n)$ which is too much for us.

3. Space Complexity: In the the space complexity analysis of Lemma 4.6.1 we used composition of space bounded algorithms, where the space complexity of each layer was roughly $O(\log w)$. However, in the iterated matrix product case there are roughly n terms in the product, and then it seems the space complexity of each layer has to be $\Omega(\log n)$ even just for the indexing. Thus, the total space complexity is $\Omega(\log^{3/2} n)$, which is again too much for us. Can we avoid this loss?

Improving the dependence on the confidence parameter

The discussion above shows that the confidence parameter has to be smaller than $\frac{1}{n}$. Nisan's generator, however, has bad dependence on δ, and this forces us to use another PRG with a better dependence on the confidence:

Lemma 5.3.1. *There exists an algorithm Λ that gets as input:*

1. *A sequence of $w \times w$ sub-stochastic matrix $(A) = (A_1, \ldots, A_n)$ in which every entry is represented using at most s bits.*

2. *An accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and a canonization parameter $t \in \mathbb{N}$, where $t \leq s$.*

3. *A seed $h \in \{0,1\}^{d_{\mathsf{N}}}$ for $d_{\mathsf{N}} = \left(\log n \cdot \left(t + \log \frac{nw}{\varepsilon}\right) + \log\log n \log \frac{1}{\delta}\right)$.*

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

$$\Lambda((A), h) = (M_h),$$

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 $1 \leqslant i \leqslant n$,

$$\|(M_h)_i - A_1 \cdots A_i\|_\infty \leq \varepsilon + nw \cdot 2^{-t}.$$

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

Comparing Lemma 5.3.1 with Lemma 4.5.4, we see that Lemma 5.3.1 improves the dependence on δ both in the the space complexity and the length of d_{N}. Henceforth, we shall denote this parameter by δ_Λ, distinguishing it from the previously used δ_{N}, and similarly ε_Λ instead of ε_{N}. The construction of Λ is as follows. We start with Nisan's PRG with *constant* confidence, and amplify its confidence to the desired δ using a *sampler*. For a formal description, and a discussion of the solution (and the use of *samplers*), see Section 5.8.2.

The space complexity of composition

As discussed above, in our case there is a subtlety regarding the space complexity of the composition, because we cannot afford to keep a fresh index at each layer of the composition. We resolve this issue by noting that some of the indices can be maintained globally. In Section 5.8.1 we prove Lemma 5.8.2 which is a generalization of the space composition theorem (Claim 2.2.2) that may be interesting on its own right.

Dealing with the shifts

The shifts are the bigger issue, and resolving it requires new ideas and technical effort. Previously, in the powering algorithm, we invested only $O(\log w)$ random bits per a single shift, and we had r_2 such shifts, one for every matrix we encounter in the computation (namely, the matrices $A, A^{2^{r_1}}, A^{2\cdot 2^{r_1}}, \dots, A^n$). However, now we have $\Omega(n)$ intermediate matrices, and we cannot afford to use a fresh shift for each intermediate matrix.

We therefore try a new approach. Instead of using a fresh shift in every iteration, we shift the input. Also, as we have n input matrices $A_1, \dots, A_n$, we use the same shift ζ on all of the n input matrices. As we need each of the shifts to work well, we need a union bound against n matrices, and we therefore use $O(\log n)$ bits for choosing the shift ζ. Thus, we cannot afford to do such a shift at each layer, and instead we study what happens when we just shift the input, and we do not shift intermediate layers. Our first attempt is the following algorithm, where $t_2 = O(\log n)$, $t_1 = O(\log w)$, $r_1 \cdot r_2 = \log n$ (and for simplicity, say $r_1 = r_2 = \sqrt{\log n}$):

1. Shift the entry of each of the input matrices by a random $\zeta \sim 2^{-2t_2} \cdot \{0, 1, \dots, 2^{t_2/2-1} - 1\}$.
2. For $i = 1, \dots, r_2$,
   (a) Partition the iterated product to sub-products, each of 2^{r_1} matrices.
   (b) Truncate the matrices to precision t_1 and use Λ to approximate the iterated sub-products.
   (c) Regain the high accuracy via the Richardson iterated, and then truncate to precision t_2.
3. Output $\widetilde{M}_{r_2}$.

As before, the role of the outer rounding is to decorrelate the randomness h from the output, and at this stage it is unclear yet whether it achieves this goal. Notice that we shift all *inputs* by the *same shift* using $O(\log n)$ bits for that single shift, whereas in the powering algorithm we used independent shifts for every intermediate calculation using $O(\log w)$ bits per shift. We stress that there are no other shifts for intermediate calculations 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 effects the output product. In Lemma 5.5.2 we prove that a shift ζ of each entry of $A_1, \dots, A_n$ results in an error matrix $E(\zeta)$, where

$$0 \leqslant E(\zeta)[i,j] \leqslant \zeta \cdot T[i,j],$$

and the matrix T is defined by:

$$T \stackrel{\text{def}}{=} J_w \sum_{k=1}^{n} A_{k+1} \cdots A_n. \tag{5.1}$$

This implies that each entry of E has magnitude at most $n^2\zeta$. However, generalizing the rounding lemma (Lemma 4.5.6) to the case where the shifts are given by some error function $E(\zeta)$ reveals that in this case we need to bound $E(\zeta)$ not only from above, but also from below. We give the precise details in Lemma 5.5.3. Luckily for us, a closer look reveals that

$$0 \leq (1 - wn\zeta)\zeta \cdot T[i,j] \leq E(\zeta)[i,j] \leq \zeta \cdot T[i,j],$$

and that with a good 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 a good probability over the shift, the truncation indeed decorrelates h from the output. We give the precise details in Section 5.5.

Furthermore, by taking a 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 *safe*, in the sense that all the entries appearing in them 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 through multiplication, but also through 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). However, 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 seemingly the argument loses $\log \frac{1}{\rho}$ bits of precision in every iteration, which is roughly $\log n$.

Our solution to the problem is to introduce another Richardson iteration step to make ρ smaller than 2^{-t_2} ultimately breaking the dependence between ρ and t_2. Specifically, our final algorithm is the following:

1. Shift the entry of each of the input matrices by a random $\zeta \sim \{0, 2^{-2t_2}, \ldots, 2^{-2t_2} \cdot (2^{t_2/2-1} - 1)\}$.
2. For $i = 1, \ldots, r_2$,
   (a) Partition the iterated product to sub-products, each of 2^{r_1} matrices.
   (b) Truncate the matrices to precision t_1 and use Λ to approximate the iterated sub-products.
   (c) Regain the high accuracy via the Richardson iterated, and then truncate to precision t_2.
   (d) Improve further the precision using the Richardson iteration to accuracy ρ.
3. Output $\widetilde{M}_{r_2}$.

An exact outline appears in the next section.

The fact that we use *two* Richardson steps at each layer may look perplexing at first. However, the utility of the two Richardson steps may be simply explained. The inner Richardson iteration, combined with the truncation performed right after, is designed to decorrelate h. On th other hand, the outer Richardson iteration maintains a small universal error ρ *independent of the inner decorrelation procedure*. Thus, while the matrix after the rounding is not safe, the outer Richardson iteration brings it closer to the correct value - so close that it must be safe.

5.4 Algorithm Outline, Correctness, and Complexity

The algorithm partitions the inputs of the iterated product into groups of size 2^{r_1}, approximates the product of elements in each group, and recurses. This defines a tree of depth $r_2 = \frac{r}{r_1}$ and arity 2^{r_1} where the $n = 2^r$ inputs are the leaves. Every level of the tree is a sequence of matrices: the bottom level consists of 2^r matrices, the level above it consists of 2^{r-r_1} matrices, and in general the i-th level consists of $2^{r-r_1 i}$ matrices[1]. Every node in the recursion tree stands for the approximated product of its children, and consequently the product of all its descendants in the tree. In order to formally outline and analyze our algorithm we shall need to find a proper way to index these nodes.

Recalling the notation of matrix sequences (see Section 4.4), let (M_i) denote the matrix sequence in level i, and consider its j-th matrix $(M_i)_j$. For simplicity, let us assume that the computation is exact, i.e., every node equals exactly the iterated product of its children. We thus have the following simple recursive relation: the product of a node's descendants, equals the product of the iterated products associated with its direct children. Algebraically, denote by $\Gamma_{i,j}$, $L_{i,j}$ the children and descendants of the j-th node in depth i then

$$\prod_{L_{i,j}}(M_0) = \prod_{k\in\Gamma_{i,j}} \prod_{L_{i-1,k}} (M_{i-1}), \tag{5.2}$$

where we use the shorthand $\prod_B(M) \overset{\text{def}}{=} \prod_{k\in B}(M)_k$. One can work out an exact formula

$$\begin{aligned}\Gamma_{i,j} &= \{k + (j-1)2^{r_1} \mid 1 \leqslant k \leqslant 2^{r_1}\},\\ L_{i,j} &= \{(j-1)2^{ir_1} + 1, (j-1)2^{ir_1} + 2, \ldots, j2^{ir_1}\}.\end{aligned}$$

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

We can now formally outline the algorithm for approximating the iterated product of stochastic matrices, and its correctness.

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

```
Algorithm: Iterated Saks–Zhou ($\mathsf{SZ_{It}}$)
    Input: Stochastic Matrices $A_1, \dots, A_n \in \mathbb{R}^{w \times w}$.
    Output: Stochastic Matrices $M \in \mathbb{R}^{w \times w}$ approximating $A_1 \cdots A_n$.
    Parameters: Set $t_1 = 4r_1 + \log w$, $t_2 = 12 \log n$. Instantiate $\Lambda$ to
    approximate powers of order $2^{r_1}$ with accuracy $\varepsilon_\Lambda = 2^{-2r_1}$, confidence $\delta_\Lambda = \frac{1}{n^3}$, and
    canonicalization parameter $t_1$. Also, set $r_1 = r_2 = \sqrt{r} = \sqrt{\log n}$.

    1. Draw $\zeta \sim \{0, 1, \dots, 2^{t_2/2-1} - 1\}$, and $h \in \{0,1\}^{d_{\mathsf{N}}}$ uniformly.
    2. Set $(\widetilde{M_0})_j = (A_j - \zeta 2^{-2t_2} J_w)$ for $j = 1, \dots, n$.
    3. For $i = 1, \dots, r_2$,
        (a) Compute
            $(\widetilde{M_i})_j = \mathsf{R}\Big(\Big\lfloor \mathsf{R}\Big(\Lambda(\lfloor (\widetilde{M}_{i-1})_{\Gamma_{i,j}} \rfloor_{t_1}, h), M, 6t_2\Big)\Big\rfloor_{t_2}, (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 8t_2\Big)$
            for $j = 1, \dots, \frac{n}{2^{ir_1}}$.
    4. Output $\widetilde{M}_{r_2}$.
```

The parameters are chosen so that they satisfy the following constraints:

$$\varepsilon_\Lambda + w \cdot 2^{r_1 - t_1} \leqslant \frac{1}{4(2^{r_1}+1)}, \text{ and,} \tag{5.3}$$

$$2^{-t_2} \leqslant \frac{1}{w(2^{r_1}+1)}. \tag{5.4}$$

so that we can apply the Richardson iteration. Also, as anyhow the algorithm only provides improvement in the regime $w \ll n$, it is harmless to assume $w \leqslant n$

Lemma 5.4.1. *For any sequence of sub-stochastic matrices* $(M) = (M_1, \dots, M_{2^r}) \in \mathbb{R}^{w \times w}$

$$\Pr_{h,\zeta}\Big[\|\mathsf{SZ_{It}}((M), h, \zeta) - M_1 \cdots M_{2^r}\|_\infty \geqslant \frac{1}{n^{12}}\Big] \leqslant \frac{2}{n^3},$$

where ζ *is chosen uniformly from* $\{0, 1, \dots, 2^{t_2/2-1} - 1\}$.

Finally, we state the space complexity of the algorithm:

Lemma 5.4.2. *The algorithm* $\mathsf{SZ_{It}}$ *can be implemented in*

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

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

Theorem 5.2.1 is proved by employing another Richardson iteration at the end in order to get better dependence on ε thus obtaining parameters identical to those in Theorem 4.2.3.

Lemma 5.4.1 is proved in Section 5.6, and Lemma 5.4.2 is proved in Section 5.7.

5.5 Perturbing the Input

We now focus on the effect of shifting the input. Suppose we want to compute the iterated product $A_1 \cdots A_n$. We shift every matrix A_i by ζ entry-wise and compute the perturbed iterated product

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

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

Definition 5.5.1. *We say $z \in \mathbb{R}$ is (t,ρ)-dangerous if z is ρ-close to a* positive *multiple of 2^{-t}, i.e., if there exists a* positive *integer $n > 0$ such that $\left|z - n2^{-t}\right| \leqslant \rho$. Otherwise, we say z is (t,ρ)-safe. Also, we say a matrix $M \in \mathbb{R}^{w\times w}$ is (t,ρ)-dangerous if one of its entries is (t,ρ)-dangerous, and otherwise we say it is (t,ρ)-safe.*

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

Remark 11. *Numbers that are very close to* 0 *are safe by definition. Indeed, negative values always round to zero, and so there are no boundary issues around* 0.

The terminology of (t,ρ)-dangerous is used in the original work of Saks and Zhou [SZ99]. We remark, however, that this terminology is not required for the Saks-Zhou algorithm as presented in Section 4.5, while it is essential for our iterated product algorithm.

In the powering algorithm of Chapter 4, analyzing the shift-and-truncate operation eventually reduced to analyzing the shift-and-truncate on numbers. In our algorithm for the iterated product, the situation is very different as we shift the input matrices, and analyze how this shifting affect the product. Let $E_{i,j}$ denote the difference between the true product and the shifted product at the $[i,j]$-th cell

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

Also, let $E(\zeta)$ be the $w \times w$ matrix, whose $[i,j]$'th entry is $E[i,j]$. The following is an explicit formula for E.

Lemma 5.5.2. *Let $A_1, \ldots, A_n \in \mathbb{R}^{w\times w}$ stochastic, and J_w the all-ones matrix. Then,*

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

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

Proof. Expanding the product, a generic term has the form,

$$A_1 \cdots A_{i_1}(-\zeta J_w)A_{i_1+2} \cdots A_{i_2}(-\zeta J_w) \cdots A_{i_j}(-\zeta J_w)A_{i_j+2} \cdots A_n.$$

For any stochastic matrix A we have that $AJ_w = J_w$, and also $J_w^j = w^{j-1}J_w$ so the above simplifies to

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

Summing over the above terms according to the last index of an all-ones matrix, and using the binomial theorem we get

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

□

It follows from Lemma 5.5.2 that

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

where T is as in eq. (5.1). Furthermore,

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

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

The next lemma is a generalization of the rounding lemma (Lemma 4.5.6) to the case where the shifts are given by some function (e.g., $E_{i,j}$), with a fairly controlled behaviour.

Lemma 5.5.3. *Let $e\colon\mathbb{R}\to\mathbb{R}$ be such that for all $\zeta\in[0,1]$*

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

where C,ℓ satisfy $0\le C<2^{t-\ell}$ for some positive integers ℓ,t. Then, for all $z\leqslant C$ we have

$$\Pr_{\zeta\in\{0,1,2,\ldots,2^{\ell}-1\}}\left[(z-e(\zeta 2^{-2t}))\ \textit{is}\ (t,2^{-3t-2})\textit{-dangerous}\right]\leqslant 2^{-\ell+1}.$$

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

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

Therefore,

$$\begin{aligned}
e((\zeta+1)2^{-2t})-e(\zeta 2^{-2t}) &\geqslant C(1-2^{-\ell-1})(\zeta+1)2^{-2t}-C\zeta 2^{-2t} \\
&= C2^{-2t}(1-2^{-\ell-1}(\zeta+1)) \\
&\geqslant C2^{-2t-1} \\
&\geqslant 2^{-3t-2}
\end{aligned}$$

where we used that $C \geqslant z \geqslant 2^{-t-1}$ and $\zeta < 2^\ell$. In other words, $e(\zeta 2^{-2t})$ is increasing with ζ and consecutive ζ-s are at least 2^{-3t-2} apart. Moreover,

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

and so there are at most two ζ-s for which $z - e(\zeta 2^{-2t})$ is $(t, 2^{-3t-2})$-dangerous. $\square$

Corollary 5.5.4. *Let $A_1, \dots, A_n \in \mathbb{R}^{w \times w}$ be stochastic matrices of dimension $w \leqslant n$. Also, let $\ell, t \in \mathbb{N}$, $\zeta \in (0,1)$ be such that $t \geqslant 4 \log n$ and $\ell < t/2$. Then*

$$\Pr_{\zeta \in \{0,1,2,\dots,2^\ell-1\}} \left[\prod_{i=1}^{n} (A_i - \zeta 2^{-2t} J_w) \textit{ is } (t, 2^{-3t-2})\textit{-dangerous} \right] \leqslant w^2 2^{-\ell+1}.$$

Proof. For each entry $[i,j]$, let

$$z = (A_1 \cdots A_n)[i,j]$$

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

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

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

$$wn\zeta 2^{-2t} < n^2 2^\ell 2^{-2t} \leq 2^{t/2} 2^\ell 2^{-2t} \leq 2^{-\ell-1}$$

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

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

Therefore, by Lemma 5.5.3, the probability over ζ that $\prod_{i=1}^{n}(A_i - \zeta 2^{-2t} J_w)[i,j]$ is $(t, 2^{-3t-2})$-dangerous, is at most $2^{-\ell+1}$. The result then follows by a union bound over all w^2 entries. $\square$

Using the fact that each $L_{i,j}$ has at most n elements, we also record the following corollary.

Corollary 5.5.5. *Let $A_1, \dots, A_n \in \mathbb{R}^{w \times w}$ be stochastic matrices of dimension $w \leqslant n$. Also, let $\ell, t \in \mathbb{N}$, $\varepsilon \in (0,1)$ be such that $t \geqslant 4 \log n$ and $\ell < t/2$. Then, for the matrix sequence $M(\zeta) = (A_1 - \zeta 2^{-2t} J_w, \dots, A_n - \zeta 2^{-2t} J_w)$*

$$\Pr_{\zeta \in \{0,1,2,\dots,2^\ell-1\}} \left[\exists i,j \prod_{L_{i,j}} (M(\zeta)) \textit{ is } (t, 2^{-3t-2})\textit{-dangerous} \right] \leqslant nw^2 2^{-\ell+1}.$$

5.6 Proof of Correctness

Proof of Lemma 5.4.1. Assume the premises of Lemma 5.4.1. Let (M) be a a sequence of sub-stochastic matrices $(M)_1, \dots, (M)_{2^r} \in \mathbb{R}^{w \times w}$. We have $t_2 \geq 4 \log n$ and $\ell < t_2/2$. Let $(M_0(\zeta))$ be the matrix sequence defined by

$$(M_0(\zeta))_i \stackrel{\text{def}}{=} (A_i - \zeta 2^{-2t_2} J_w),$$

for $i = 1, \ldots, n$. By Corollary 5.5.5, except for probability $nw^2 2^{-t_2/2+2}$, for all i, j, $\prod_{L_{i,j}}(M_0(\zeta))$ is $(t_2, 2^{-3t_2-2})$-safe. Now recursively define

$$(M_i(\zeta))_j \overset{\text{def}}{=} \mathsf{R}\left(\left\lfloor \prod_{\Gamma_{i,j}}(M_{i-1}(\zeta))\right\rfloor_{t_2}, (M_{i-1}(\zeta))_{\Gamma_{i,j}}, 8t_2\right). \tag{5.5}$$

Henceforth, we shall assume that ζ is "good" i.e., all sub-products

$$\prod_{L_{i,j}}(M_0(\zeta))$$

are $(t_2, 2^{-3t_2-2})$-safe. Also, for brevity we shall omit the dependence of (M_i), $(\widetilde{M_i})$ in ζ, and that of (M_i) in h. The proof essentially follows from the next two claims, which are very similar to those in Section 4.6.

Claim 5.6.1. *For all i, j $\left\|(M_i(\zeta))_j - \prod_{L_{i,j}}(M_0(\zeta))\right\|_\infty \leqslant 2^{r_1-7t_2}$.*

With the above claim in hand we can prove the following claim.

Claim 5.6.2. $\Pr_h\left[\exists i,\ (M_i) \neq (\widetilde{M_i})\right] \leqslant n\delta_\Lambda$.

We therefore see that except for probability $n\delta_\Lambda$, the sequences $(\widetilde{M_i}(\zeta))$ that the algorithm constructs are identical to the sequences $(M_i(\zeta))$. By Claim 5.6.1, and Claim 2.4.5

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

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

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

Using the triangle inequality and taking $i = r_2$ gives

$$\Pr_h\left[\|\mathsf{SZ}_{\mathsf{It}}((M), h) - M_1 \cdots M_{2^r}\|_\infty \geqslant 2^{-t_2}\right] \leqslant n(w^2 2^{-t_2/2+2} + \delta_\Lambda).$$

The parameters of Lemma 5.4.1 follows from our choice of parameters. □

Proof of sub-lemmas

Proof of Claim 5.6.1. We prove by induction on i that

$$\left\|(M_i(\zeta))_j - \prod_{L_{i,j}}(M_0)\right\|_\infty \leqslant 2^{r_1-8t_2}(2^{ir_1} - 1)$$

for all $j = 1, \ldots, \frac{n}{2^{ir_1}}$. The base case $i = 0$ is trivial. By the induction hypothesis we get that for every k

$$\left\|(M_{i-1}(\zeta))_k - \prod_{L_{i-1,k}}(M_0)\right\|_\infty \leqslant 2^{r_1-8t_2}(2^{r_1(i-1)} - 1)$$

for all $j = 1, \ldots, \frac{n}{2^{(i-1)r_1}}$. Using Equation (5.2) and the induction hypothesis

$$\left\| \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) - \prod_{k \in \Gamma_{i,j}} \prod_{L_{i-1,k}} (M_0) \right\|_\infty \leqslant 2^{r_1} \cdot 2^{r_1 - 8t_2}(2^{r_1(i-1)} - 1)$$

for all $j = 1, \ldots, 2^{r_1}$. The same bound also holds if we consider only a sub-sequence of $\Gamma_{i,j}$. Also note that

$$\left\| \lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) \rfloor_{t_2} - \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) \right\|_\infty \leqslant w 2^{-t_2+1}.$$

Thus, by the Richardson iteration Lemma 4.4.1 and our choice of parameters (Equation (5.3), Equation (5.4)),

$$\left\| \mathsf{R}(\lfloor \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) \rfloor_{t_2}, (M_{i-1}(\zeta))_{\Gamma_{i,j}}, 8t_2) - \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) \right\|_\infty \leqslant 2^{r_1 - 8t_2},$$

and so by the definition of $(M_i(\zeta))_j$

$$\left\| (M_i(\zeta))_j - \prod_{\Gamma_{i,j}} (M_{i-1}(\zeta)) \right\|_\infty \leqslant 2^{r_1 - 8t_2}.$$

Putting it altogether

$$\begin{aligned} \left\| (M_i(\zeta))_j - \prod_{L_{i,j}} (M_0) \right\|_\infty &\leqslant 2^{r_1 - 8t_2} + 2^{r_1 - 8t_2} \cdot 2^{r_1} \cdot (2^{2r_1(i-1)} - 1) \\ &\leqslant 2^{r_1 - 8t_2}(2^{r_1(i-1)} - 1). \end{aligned}$$

This completes the induction. The assertion follows from the observation that $ir_1 \leqslant \log n \leqslant t_2$, and our choice of parameters. $\square$

Proof of Claim 5.6.2. We prove by induction on i that

$$\Pr_h \left[\exists k \leqslant i, \ (M_k) \neq (\widetilde{M_k}) \right] \leqslant \varepsilon_i \overset{\text{def}}{=} (2^{r - r_1} - 2^{r - r_1(i+1)})\delta_\Lambda.$$

The base case $i = 0$ is trivial. Fix some $i \geq 1$, assume the induction holds for $i - 1$ and prove for i. By the induction hypothesis,

$$\Pr_h \left[\exists k \leqslant i \ (M_k) \neq (\widetilde{M_k}) \right] \leqslant \varepsilon_{i-1} + \Pr_h \left[\exists j \ (M_i)_j \neq (\widetilde{M_i})_j \mid (M_{i-1}) = (\widetilde{M_{i-1}}) \right].$$

Thus, by the union bound it suffices to show that for every $1 \leqslant j \leqslant 2^{r - r_1 i}$

$$\Pr_h \left[(M_i)_j \neq (\widetilde{M_i})_j \mid (M_{i-1}) = (\widetilde{M_{i-1}}) \right] \leqslant \delta_\Lambda.$$

Fix j and assume that the conditioning holds i.e., $(M_{i-1}) = (\widetilde{M_{i-1}})$. Using Lemma 4.4.1 and our choice of parameters (Equation (5.3), Equation (5.4))

$$\Pr_h \left[\left\| \mathsf{R}\Big(\Lambda(\lfloor (M_{i-1})_{\Gamma_{i,j}} \rfloor_{t_1}, h), (\widetilde{M_{i-1}})_{\Gamma_{i,j}}, 6t_2\Big) - \prod_{\Gamma_{i,j}} (M_{i-1}) \right\|_\infty \geqslant 2^{-5t_2} \right] \leqslant \delta_\Lambda.$$

Indeed, we shall prove that if the above does not hold

$$\left\| \mathsf{R}\Big(\Lambda(\lfloor(M_{i-1})\rfloor_{t_1}, h), (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 6t_2\Big) - \prod_{\Gamma_{i,j}}(M_{i-1}) \right\|_\infty \leqslant 2^{-5t_2}, \tag{5.6}$$

then $(\widetilde{M}_i)_j = (M_i)_j$. Inspecting the definitions of (M_i), $(\widetilde{M}_i)$ we can see that as already $(M_{i-1}) = (\widetilde{M}_{i-1})$ then it is enough to show that

$$\lfloor \prod_{\Gamma_{i,j}}(M_{i-1})\rfloor_{t_2} = \lfloor \mathsf{R}\Big(\Lambda(\lfloor(M_{i-1})\rfloor_{t_1}, h), (\widetilde{M}_{i-1})_{\Gamma_{i,j}}, 6t_2\Big)\rfloor_{t_2}. \tag{5.7}$$

(For the definitions see Equation (5.5) and the $\mathsf{SZ}_{\mathsf{It}}$ algorithm in Section 5.3). Observe that

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

The first equality is due to the conditioning $(M_{i-1}) = (\widetilde{M}_{i-1})$, the second equality due to Equation (5.2), the third inequality is due to Claim 2.4.5, and the last inequality is due to the induction hypothesis and that $r_1 \leqslant t_2$.

Recall that ζ is such that the product

$$\prod_{L_{i,j}}(M_0)$$

is $(t_2, 2^{-3t_2-2})$-safe. Alternatively, its entries are at least 2^{-3t_2-2} far away from the boundary of truncation by t_2 bits. Using that

$$\left\| \prod_{\Gamma_{i,j}}(\widetilde{M}_{i-1}) - \prod_{L_{i,j}}(M_0) \right\|_{\max} \leqslant \left\| \prod_{\Gamma_{i,j}}(\widetilde{M}_{i-1}) - \prod_{L_{i,j}}(M_0) \right\|_\infty \leqslant 2^{-6t_2}$$

we conclude that the product

$$\prod_{\Gamma_{i,j}}(M_{i-1})$$

is also safe, specifically it is (t_2, ρ')-safe for $\rho' = 2^{-3t_2-2} - 2^{-6t_2}$. Since $\rho' > 2^{-5t_2}$ then Equation (5.6) implies Equation (5.7). This completes the inductive step. $\square$

5.7 The Space Complexity

We now analyze the space complexity of the algorithm. As a start, let us try to employ the same approach as in Section 4.6. Define the functions f_i that take a sequence (M) of length n and output the sequence

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

which is exactly the i-th iteration in the $\mathsf{SZ}_{\mathsf{It}}$ algorithm. Clearly, the composition

$$f_{r_2} \circ \cdots \circ f_1((M))$$

computes "row by row" the output of the iterated Saks–Zhou algorithm on an input (M). The problem is that each f_i requires at least

$$r - ir_1 = \log n - i\sqrt{\log n}$$

space just for indexing as j runs from 1 to 2^{r-ir_1}, and this accumulates to

$$\sum_i \log n - i\sqrt{\log n} \geqslant \frac{\log^{3/2} n}{4}$$

space. However, there is a redundancy in the above approach: each f_i maintains a global index to its location in the recursion tree, and to get around this we will globally store that index. Each node, computes the local function that takes a sequence (M) of length r_1 and outputs

$$\mathsf{R}\Big(\big\lfloor \mathsf{R}\big(\Lambda(\lfloor (M)_{\Gamma_{i,j}} \rfloor_{t_1}, h), M, 6t_2\big)\big\rfloor_{t_2}, (M)_{\Gamma_{i,j}}, 8t_2\Big),$$

where it is given its children $(M)_{\Gamma_{i,j}}$ as input. The above function only maintains indices locally, namely it does not know its location within the recursion. Knowing the local indices for every level of the recursion in tandem with the global location suffices for the algorithm to operate. This framework is formalized in Lemma 5.8.2.

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

$$(\widetilde{M_i})_j = f((M_{i-1})_1, \ldots, (M_{i-1})_{2^{\sqrt{\log n}}})$$

(See the $\mathsf{SZ}_{\mathsf{It}}$ algorithm in Section 5.3). As a special case, Lemma 5.8.2 states that any algorithm of that sort - a tree in which every node is some function f of its children, can be implemented in space that it takes to compute f times the depth of the tree. According to Lemma 5.3.1 and Lemma 4.4.1, the function f can be computed via

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

and so Lemma 5.4.2 follows.

5.7.1 On Oblivious Approximations

Our algorithms for randomized exponentiation/product (Lemma 4.5.4, Lemma 5.3.1) has a special property that enables a more space-efficient implementation - *obliviousness*. For the simplicity of the presentation, it is better to first think about the input as an ROBP, rather than a matrix, ignoring the canonicalization procedure. The algorithm Λ has the following special structure:

1. The algorithm Λ computes a collection of sequences $I(h) \subseteq \Sigma^n$ depending on the randomness y.

2. Every sequence $\sigma \in I(h)$ corresponds to the Boolean stochastic matrix $B(\sigma)$.

3. The matrices $B(\sigma)$ are aggregated to a single matrix via the function g (e.g., g takes the average over all sequences, weighted average, median-of-averages, etc.).

For instance, in Lemma 4.5.4 the sequences $I(h)$ are generated via hash functions and g is the averaging function, and in Lemma 5.3.1 the sequences $I(h)$ are generated via random walks on expander graph and g is the median-of-averages. In the setting of $\mathsf{SZ}_{\mathsf{It}}$, the input (and output) of Λ is a matrix, and not an ROBP. However, this makes no difference as the input matrix is used to build a canonical ROBP.

Here, obliviousness refers to the fact that the sequences $I(h)$ are universal, and do not depend on B. This enables us to pre-compute the sequences $I(h)$, which is usually the more difficult task computationally, and so every iteration only simulates $B(\sigma)$ for every $\sigma \in I(h)$. It is possible to implement the $\mathsf{SZ}_{\mathsf{It}}$ algorithm so that the contribution to the space complexity is

$$S_I + r_2 \cdot (S_g + \log |I(h)|)$$

where S_I is the space complexity required to compute $I(h)$, and S_g is the space complexity of g. Note that S_I is not multiplied by r_2. One way to see that, is using Claim 2.2.2 which enables us to assume that $I(h)$ is given as part of the input instead of h. Another (equivalent) way is via the framework of Lemma 5.8.2: viewing $I(h)$ as a **single** node in the algorithm's layout used by the other nodes.

While in our implementation this saving is not acute, it might be a good idea to have that saving in mind. A good example in which this observation does make a difference is in Hoza's improvement [Hoz21] to the Saks–Zhou algorithm, where he obtained a space complexity of

$$\frac{\log^{3/2} n}{\sqrt{\log\log n}}$$

in the case $w = n$, and $\varepsilon = \frac{1}{n}$. The reason this observation is crucial in his argument is because he uses Armoni's generator [Arm98], which roughly shaves a $\log\log w$ factor from the **seed length** of the INW generator (see Section 2.7.2). While Armoni's seed length is slightly shortened, its provable space complexity is linear in the seed length. Even if the space complexity were to be improved significantly, then it is still likely to have a doubly-logarithmic factor (e.g., as in the space complexity of the INW generator Section 2.7.2) - dominating the improvement in the seed length. However, if one pre-computes the sequences produced by Armoni's generator then its inferior space complexity does not affect the overall space complexity of the Saks–Zhou algorithm.

Remark 12. *One can perhaps use Armoni's generator as Hoza did to obtain a* $\log\log$ *improvement, but we have not verified it.*

5.8 Appendix

5.8.1 Improved Space Bounded Composition

We have seen the space composition theorem (Claim 2.2.2), where the space complexity of the composition is the sum of the space complexities of each separate layer. However, sometimes the composition can be implemented in a cheaper way. A notable example for that is the $\mathbf{NC}^1 \subseteq \mathbf{L}$ inclusion. An $\mathbf{NC}^1$ circuit has depth $O(\log n)$ and elementary Boolean functions as gates (And, Or and Negation). Applying Claim 2.2.2 we get a simulation in $O(\log^2 n)$ space, because each of the $O(\log n)$ layers requires $O(\log n)$ bits just for keeping an index into its input. Looking at the $\mathbf{NC}^1 \subseteq \mathbf{L}$ proof, one sees that the cheaper simulation maintains indexing in a *global* way, thus avoiding the need to pay an index at each layer. This global indexing is made possible because the computation has a *tree* structure.

In this section we prove Lemma 5.8.2 which is a natural combination of the above two fundamental building blocks in space-bounded computation i.e.,

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

Definition 5.8.1. *Let G be a set of functions $g\colon \{0,1\}^\star \to \{0,1\}^\star$ such that that each $g \in G$ can be computed by a TM. A generalized Boolean formula F with gates in G over the input variables $x_1, \dots, x_n$ is a labeled directed a-cyclic graph such that:*

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

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

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

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

Two spacial cases to bear in mind are:

- The case where the graph has $O(\log n)$ depth and G is the set of elementary Boolean functions (And, Or, Negation), which corresponds to an $\mathbf{NC}^1$ computation, and,
- The case where the graph is a directed path and G is the set of functions in $\mathbf{L}$, which corresponds to a general composition of space bounded algorithms.

Lemma 5.8.2. *Suppose $f(x_1, \dots, x_n)$ can be computed by a generalized formula with gates in G. Furthermore,*

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

Then, we can compute the function f in

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

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

In addition, it is possible to apply Lemma 5.8.2 for arbitrary a-cyclic graphs (a.k.a. circuits) by converting them to a formula, paying an additive term of $O(\log|\mathcal{P}|)$ in the space complexity. For example, for $\mathbf{NC}^1$, $S = O(\log n)$ and $s(v_i) = \ell(v_i) = O(1)$.

The proof is essentially a combination of the above ingredients so we only highlight needed modifications.

Proof Sketch. As in the space-bounded algorithm for Boolean formulas, we traverse the graph bottom-up, namely starting from the output gates, recursively computing the values of the children:

1. Maintain globally the current node within the formula. This is stored at the beginning of the work-tape.

2. As in the space composition algorithm (Claim 2.2.2), we maintain a stack for every level of the recursion.

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

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

   i. If this is an input gate, this can be read from the input-tape.

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

□

5.8.2 Improving the Confidence Parameter

We now discuss the proof of Lemma 5.3.1, and related concepts.

Let us start by giving a different perspective on the Nisan generator (See Section 2.7.1). Recall that the Nisan generator has two types of inputs: a symbol $\sigma \in \Sigma$, and a sequence of hash functions $h = (h_1, \dots, h_k)$. Define the collection of functions

$$\{G_h(x) \overset{\text{def}}{=} \mathsf{N}(x,h) \mid h \in \mathcal{H}^k\}. \tag{5.8}$$

Observe that each G_h is a function that takes

$$s = O\left(\log \frac{nw|\Sigma|}{\varepsilon_{\mathsf{N}}\delta_{\mathsf{N}}}\right)$$

bits, and outputs n symbols from Σ using $O(s)$ space. Another way to interpret Theorem 2.7.1 is that for a predetermined ROBP if we *sample* a function from the collection (5.8) then with high probability it fools that ROBP with accuracy ε_{N}. This special "sampling property" of the

Nisan generator was used to derive two of the most important derandomization results of **BPL** [Nis94, SZ99].

In hindsight, the aforementioned special property of Nisan's generator can be obtained generically using a primitive called *averaging samplers*. By "generically" we mean that there is a simple procedure that endows a given PRG with this "sampling property" (See Lemma 5.8.4). The idea of using samplers in the context of PRGs against the class of ROBPs is not new, and dates back to the work of Armoni [Arm98]. Moreover, samplers played a crucial role in [BCG20, CL20]. While the technical statements are by no means new, the following consequence eluded previous works.

We now introduce the notion of samplers, and briefly explain how to derive Lemma 5.3.1. The presentation mainly follows the highly recommended survey [Gol97].

Definition 5.8.3. *A sampler with accuracy ε, and sampling error δ, is a randomized algorithm ν^f with oracle access to a function $f\colon \{0,1\}^n \to [0,1]$ such that*

$$\forall f \;\Pr_y\left[\left|\nu^f(y) - \mathbb{E}_\sigma[f(\sigma)]\right|\right] \geqslant \epsilon] \leqslant \delta.$$

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

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

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

iii. Randomness: Amount of randomness used by the sampler.

A very important type of samplers is that of *averaging samplers*, in which the sampler simply outputs the average of its queries. An averaging sampler is thus defined via a function

$$S\colon \{0,1\}^d \times \{0,1\}^m \to \{0,1\}^n.$$

The output of ν^f is then

$$\nu^f(y) = \mathbb{E}_{x\in\{0,1\}^d}[f(S(x,y))],$$

and so the query and randomness complexity of the above sampler are m, 2^d respectively. More generally, and analogously to the discussion on oblivious approximations in Section 5.7, one can consider non-adaptive samplers in which

$$\nu^f(y) = g((f(S(x,y))_{x\in\{0,1\}^d}),$$

where g is an arbitrary function. (For averaging samplers g is the average function.).

Let us differ the discussion about the existence of efficient samplers, and go back to pseudorandom generators. Suppose that we have a PRG $G\colon \{0,1\}^s \to \Sigma^n$ against $\mathrm{B}[n,w,\Sigma]$, i.e., for every ROBP B

$$\|\mathbb{E}_x[B(G(x))] - \mathbb{E}_\sigma[B(\sigma)]\|_\infty \leqslant \varepsilon.$$

Note that G does not necessarily satisfy the guarantee of the Nisan generator discussed above. E.g., G can be taken to be the INW generator (See Section 2.7.2). Set the function

$$f(x) \stackrel{\text{def}}{=} \mathbb{E}_x[B(G(x))],$$

then it natural to try and sample from f using a sampler. Consider

$$G_S \stackrel{\text{def}}{=} G \circ S, \tag{5.9}$$

namely $G_S(x,y) = G(S(x,y))$. It is worth pointing out that if S is an averaging sample then G_S is a PRG. Using a non-averaging sampler ν^f we would have obtained a randomized **algorithm** that approximates the transition matrix in a black-box fashion (i.e., only queries the ROBP as a function), and for non-adaptive samplers the approximation is "oblivious".

Lemma 5.8.4. *Let $S\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ be an averaging sampler with accuracy ϵ_S, and sampling error δ_G. Also, let $G\colon \{0,1\}^s \to \{0,1\}^n$ be a PRG against $\mathrm{B}[n,w,\Sigma]$ with accuracy ε_G. Then for every $B \in ([w] \times \Sigma \to [w])$*

$$\Pr_x\left[\|\mathbb{E}_y[B(G_S(x,y))] - \mathbb{E}_\sigma[B(\sigma)]\|_{\max} \geqslant \varepsilon_G + \varepsilon_S\right] \leqslant w^2\delta_S.$$

Moreover, G_S can be implemented in the space required to implement S, plus the space required to implement G.

Proof of Lemma 5.8.4. For every indices $1 \leqslant i,j \leqslant w$ consider the Boolean function

$$f_{i,j}(z) \overset{\text{def}}{=} (B(G(z))[i,j].$$

By the sampler property

$$\Pr_x[|(B(G(S(x,y)))[i,j] - (\mathbb{E}_z[B(G(z))])[i,j]| \geqslant \varepsilon_S] \leqslant \delta_S.$$

By union bound the above holds for all indices $1 \leqslant i,j \leqslant w$ expect with probability $w^2\delta_S$ and so

$$\Pr_x[\|(B(G(S(x,y))) - \mathbb{E}_z[B(G(z))]\| \geqslant \varepsilon_S] \leqslant w^2\delta_S.$$

Using that

$$\|\mathbb{E}_z[B(G(z))] - \mathbb{E}_\sigma[B(\sigma)]\|_{\max} \leqslant \varepsilon_G.$$

completes the proof. □

It is time to discuss explicit constructions and parameters. The following is a well-known construction of an averaging sampler with very good parameters.

Theorem 5.8.5 (Corollary 7.3 in [RVW02])**.** *There exists an averaging (ϵ,δ)-Sampler $S\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ with $m = n + \log(1/\varepsilon\delta)$ and $d = O(\log(1/\epsilon) + \log\log(1/\delta))$, assuming*

$$\log(1/\delta) > \log(1/\varepsilon)2^{O(\log^*(1/\delta))}[2].$$

The space requirement of Theorem 5.8.5 are implicit in the literature, though inspecting the construction one sees that it uses simple primitives such as hash functions, expander graphs, and k-wise independent distributions - all known to have constructions that run in logarithmic space. Therefore, it is claimed that the sampler of Theorem 5.8.5 can be implemented in $O(m)$. By the observation made at the end of Lemma 5.4.2, such space complexity suffices for the $\mathsf{SZ}_{\mathsf{It}}$ algorithm. Still, as the sampler of Theorem 5.8.5 is involved and it is preferable to use an alternative simpler construction even at the cost of sub-optimality. Such an alternative route does exist, though it does take a toll: it is not an averaging sampler.

[2] $\log^* n$ denotes the iterated logarithm.

Lemma 5.8.6 (Median-of-Averages sampler [BGG93])**.** *There exists an averaging (ϵ, δ)-Sampler $S\colon \{0,1\}^m \times \{0,1\}^d \to \{0,1\}^n$ with the following parameters:*

$$m = n + \log(1/\varepsilon\delta),\ d = O(\log\frac{1}{\varepsilon} + \log\log n \log\log\frac{1}{\delta}),$$

that runs in $O(\log\log\frac{n}{\varepsilon} + \log\log\frac{1}{\delta})$ space.

The above sampler is very simple, and so we sketch its construction and briefly analyze its space complexity. There are two ingredients:

1. Pairwise independent distribution over $(\{0,1\}^n)^m$ with $m = O\big(\frac{1}{\varepsilon\delta}\big)$: This can be implemented via taking linear combinations in $\mathbb{F}_{2^n}$, and can be sampled using $r = 2n$ bits. Recall that arithmetic operations over $\mathbb{F}_{2^n}$ can be done in logarithmic space (e.g.[HAB02, HV06]).

2. Expander graphs over the vertex set $\{0,1\}^r$ with degree $D \overset{\text{def}}{=} \log n$, and spectral gap λ such that $\lambda/D < 0.1$. We use the expander graph of Section 2.7.2 which is a Cayley graph of an Abelian group so that random walks can be computed space-efficiently. That is, given $\gamma_1, \dots, \gamma_\ell \in \{1, \dots, D\}^\ell$, can be computed in space $O(\log\ell + \log\log D)$.

Let $f\colon \{0,1\}^n \to [0,1]$ be some bounded function. The sampler takes a random walk on the expander graph of length $\ell = O(\log\frac{1}{\delta})$, uses the vertices along the path to obtain ℓ samplers $(Z_1^{(j)}, \dots, Z_m^{(j)})_{j=1}^\ell$ of the pairwise independent distribution, and samples the points $Z_i^{(j)} \in \{0,1\}^n$. We then take the average over every batch

$$\mu_j = \mathbb{E}_i[f(Z_i^{(j)})],$$

and output the median of those averages median$(\mu_1, \dots, \mu_\ell)$. A detailed analysis is provided in [Gol97]. The space complexity follows from the above assertions, and composition of space bounded algorithms (Claim 2.2.2).

Remark 13. *Taking the median in Section 4.8 is in fact a naive implementation of the median-of-averages sampler: it takes the median of independent samples, whereas the median-of-averages sampler takes the median of the dependent samplers obtained by taking a random walk over an expander graph.*

Lemma 5.3.1 follows almost directly from instantiating Lemma 5.8.4 with the median-of-averages sampler of Lemma 5.8.6.

Bibliography

[AB09a] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[AB09b] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 2009.

[ACGP21] Emmanuel Abbe, Elisabetta Cornacchia, Yuzhou Gu, and Yury Polyanskiy. Stochastic block model entropy and broadcasting on trees with survey. In *Conference on Learning Theory*, pages 1–25. PMLR, 2021.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[AHN20] Emmanuel Abbe, Jan Hązła, and Ido Nachum. Almost–reed–muller codes achieve constant rates for random errors. arXiv:2004.09590, 2020.

[AKM$^+$09] Erdal Arıkan, Haesik Kim, Garik Markarian, U Ozgur, and Efecan Poyraz. Performance of short polar codes under ml decoding. *Proc. ICT MobileSummit,(Santander, Spain)*, pages 10–12, 2009.

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

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[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.

[Ar 09] Erdal Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.

[ASW15] Emmanuel Abbe, Amir Shpilka, and Avi Wigderson. Reed–muller codes for random erasures and errors. *IEEE Transactions on Information Theory*, 61(10):5229–5252, 2015.

[ASY20] Emmanuel Abbe, Amir Shpilka, and Min Ye. Reed–muller codes: Theory and algorithms. *IEEE Transactions on Information Theory*, 2020.

[BCD$^+$89] Allan Borodin, Stephen A Cook, Patrick W Dymond, Walter L Ruzzo, and Martin Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on computing*, 18(3):559–578, 1989.

[BCG20] Mark Braverman, Gil Cohen, and Sumegha Garg. Pseudorandom pseudo-distributions with near-optimal error for read-once branching programs. *SIAM Journal on Computing*, 49(5):STOC18–242–STOC18–299, 2020.

[BCP83] Allan Borodin, Stephen Cook, and Nicholas Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1-3):113–136, 1983.

[Ber84] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147 – 150, 1984. URL: http://www.sciencedirect.com/science/article/pii/0020019084900188, doi:http://dx.doi.org/10.1016/0020-0190(84)90018-8.

[BFNW93] Lźszló Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. Bpp has subexponential time simulations unlessexptime has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

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

[BK97] Jean Bourgain and Gil Kalai. Influences of variables and threshold intervals under group symmetries. *Geometric & Functional Analysis GAFA*, 7(3):438–461, 1997.

[CCL$^+$15] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Spectral sparsification of random-walk matrix polynomials. *arXiv preprint arXiv:1502.03496*, 2015.

[CDL99] A Chiu, G Davida, and B Litow. Nc1 division. *Manuscript, November*, 1999.

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

[CDSTS22] Gil Cohen, Dean Doron, Ori Sberlo, and Amnon Ta-Shma. Approximating large powers of stochastic matrices in small space. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2022. Manuscript.

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

[CKP$^+$16] Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*. IEEE, 2016.

[CKP$^+$17] Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*. ACM, 2017.

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

[Coo85] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1- 3):2–22, 1985. International Conference on Foundations of Computation Theory.

[Csa76] L. Csansky. Fast parallel matrix inversion algorithms. *SIAM Journal of Computing*, 5(6):618–623, 1976.

[DGMW70] Philippe Delsarte, Jean-Marie Goethals, and F Jessie Mac Williams. On generalized reed–muller codes and their relatives. *Information and Control*, 16(5):403–442, 1970.

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

[EKL19] David Ellis, Nathan Keller, and Noam Lifshitz. Stability versions of erdős–ko–rado type theorems via isoperimetry. *Journal of the European Mathematical Society*, 21(12):3857–3902, 2019.

[FK96] Ehud Friedgut and Gil Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996.

[FR21] Bill Fefferman and Zachary Remscrim. Eliminating intermediate measurements in space-bounded quantum computation. In *53rd Annual Symposium on Theory of Computing (STOC 2021)*, pages 1343–1356. ACM, 2021.

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

[Gol97] Oded Goldreich. A sample of samplers: A computational perspective on sampling. *def*, 1:2n, 1997.

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

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

[GWW07] Miguel Griot, Wen-Yen Weng, and Richard D Wesel. A tighter bhattacharyya bound for decoding error probability. *IEEE Communications Letters*, 11(4):346–347, 2007.

[HAB02] William Hesse, Eric Allender, and David A Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[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.

[Hoz19] William M Hoza. Typically-correct derandomization for small time and space. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[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.

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

[HSS21] Jan Hązła, Alex Samorodnitsky, and Ori Sberlo. On codes decoding a constant fraction of errors on the bsc. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1479–1488, 2021.

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

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

[HZ20] William M. Hoza and David Zuckerman. Simple optimal hitting sets for small-success **RL**. *SIAM Journal on Computing*, 49(4):811–820, 2020.

[IKW02] Russel Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

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

[IW94] Russell Impagliazzo and Avi Wigderson. $\mathbf{P} = \mathbf{BPP}$ if $\mathbf{E}$ requires exponential circuits: Derandomizing the XOR lemma. In *29th Annual Symposium on Theory of Computing (STOC 1994)*, pages 220–229. ACM, 1994.

[Jun81] H Jung. Relationships between probabilistic and deterministic tape complexity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 339–346. Springer, 1981.

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity*, 13(1-2):1–46, 2004.

[KKM$^+$16] Shrinivas Kudekar, Santhosh Kumar, Marco Mondelli, Henry D Pfister, and Rüdiger Urbankez. Comparing the bit-map and block-map decoding thresholds of reed-muller codes on bms channels. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1755–1759. Ieee, 2016.

[KKM$^+$17] Shrinivas Kudekar, Santhosh Kumar, Marco Mondelli, Henry D Pfister, Eren Şaşoğlu, and Rüdiger L Urbanke. Reed–muller codes achieve capacity on erasure channels. *IEEE Transactions on information theory*, 63(7):4298–4316, 2017.

[KL97] Ilia Krasikov and Simon Litsyn. Estimates for the range of binomiality in codes' spectra. *IEEE Transactions on Information Theory*, 43(3):987–991, 1997.

[KLP12] Tali Kaufman, Shachar Lovett, and Ely Porat. Weight distribution and list-decoding size of reed–muller codes. *IEEE Transactions on Information Theory*, 58(5):2689–2696, 2012.

[Mac99] David JC MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

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

[MRSV17] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil Vadhan. Derandomization beyond connectivity: Undirected laplacian systems in nearly logarithmic space. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 801–812. IEEE, 2017.

[MRSV19] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil Vadhan. Deterministic approximation of random walks in small space. *arXiv preprint arXiv:1903.06361*, 2019.

[Mul54] David E Muller. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the IRE Professional Group on Electronic Computers*, EC-3(3):6–12, 1954.

[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.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

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

[Oxl06] James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

[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.

[Rab80] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[Ree54] I. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

[Ros05] Raphaël Rossignol. Threshold for monotone symmetric properties through a logarithmic sobolev inequality. *The Annals of Probability*, 34(5):1707–1725, 2005.

[RP21] Galen Reeves and Henry D Pfister. Reed-muller codes achieve capacity on bms channels. *arXiv preprint arXiv:2110.14631*, 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.

[RTV06] Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 457–466. ACM, 2006.

[RU08] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory.* Cambridge University Press, 2008.

[RV05] Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2005)*, pages 436–447. Springer, 2005.

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

[Sak96] Michael Saks. Randomization and derandomization in space-bounded computation. In *Proceedings of Computational Complexity (Formerly Structure in Complexity Theory)*, pages 128–149. IEEE, 1996.

[Sam19] Alex Samorodnitsky. An upper bound on ℓ_q norms of noisy functions. *IEEE Transactions on Information Theory*, 66(2):742–748, 2019.

[Sam20] Alex Samorodnitsky. An improved bound on ℓ_q norms of noisy functions. arXiv:2010.02721, 2020.

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

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[SS77] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM journal on Computing*, 6(1):84–85, 1977.

[SS20] Ori Sberlo and Amir Shpilka. On the performance of reed-muller codes with respect to random errors and erasures. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1357–1376. SIAM, 2020.

[SY10] Amir Shpilka and Amir Yehudayoff. *Arithmetic circuits: A survey of recent results and open questions.* Now Publishers Inc, 2010.

[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 Sceinces*, 58(2):376–403, 1999.

[TB99] Stephan Ten Brink. Convergence of iterative decoding. *Electronics letters*, 35(10):806–808, 1999.

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

[TZ00] Jean-Pierre Tillich and Gilles Zémor. Discrete isoperimetric inequalities and the probability of a decoding error. *Combinatorics Probability and Computing*, 9(5):465–479, 2000.

לידי ביטוי במקרים שבהם $S \ll \log T$, ובפרט רק תקף לאלגוריתמים שבהם זמן הריצה גדול משמעותית מהזיכרון שהם צורכים. אלגוריתמים כאלה בהכרח אינם עוצרים (לעיתים), ולכן חשוב להדגיש כי אנחנו מאפשרים לאלגוריתם לא לעצור בהסתברות זניחה ודורשים כי זמן הריצה הוא בתוחלת T.

דרך שקולה לנסח את הבעיה של דה-רנדומיזציה של אלגוריתמים מוגבלי-זיכרון היא באמצעות הבעיה של כפל מטריצות. למעשה, התוצאה של סאקס וז׳ו מנוסחת כאלגוריתם יעיל לקירוב חזקות של מטריצות סטוכסטיות. אם להיות יותר מדויק, חישוב אקראי שרץ בזמן T וזיכרון S פחות או יותר שקול לבעיה של קירוב את החזקה ה-T של מטריצה סטוכסטית ממימד 2^S. לכן, טבעי לשאול באופן כללי כמה זיכרון נדרש כדי לקרב מכפלה של מטריצות סטוכסטיות נתונות

$$A_1, \dots, A_n \in \mathbb{R}^{w \times w}.$$

האלגוריתם היעיל ביותר שהיה ידוע לבעיה היה האלגוריתם לעיל שרץ בזיכרון

$$O(\log^{3/2} n + \sqrt{\log n} \log w).$$

בעבודה [CDSTS22] אנחנו מתארים אלגוריתם לבעיה זו שרץ בערך בזיכרון

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

כלומר משפר את סיבוכיות הזיכרון בתחום $n \ll w$. כדי להדגים את ההבדל בפרמטרים נתבונן במקרה שבו

$$w = 2^{\sqrt{\log n}},$$

קרי יש לנו n מטריצות ממימד $2^{\sqrt{\log n}}$. האלגוריתם של סאקס וז׳ו רץ בזיכרון של $O(\log^{3/2} n)$ עבור פרמטרים אלו, ולמעשה ניתן להשיג אלגוריתם עם פרמטרים זהים בקלות ללא שימוש באלגוריתם של סאקס וז׳ו. לעומת זאת, האלגוריתם שלנו משיג סיבוכיות זיכרון כמעט מיטבית של $O(\log n \log \log n)$.

נעיר כי כאשר כמות הזיכרון קטנה משמעותית מאורך הפלט או הקלט, נדרש להקצות למכונה שלושה סרטים: סרט קלט עליו כתוב הקלט וממנו למכונה מותר רק לקרוא, סרט פלט אליו כותבת המכונה את הפלט ונמנע ממנה לקרוא ממנו, וסרט עבודה בו המכונה משתמשת כדי לבצע את המשימה החישובית. הזיכרון מוגדר להיות אורך **סרט העבודה** שהמכונה משתמשת בו בפועל.

נעבור לדבר על משאב האקראיות. חישוב הסתברותי הינו חישוב שמשתמש במטבעות אקראיים. להבדיל מחישוב דטרמיניסטי, אנחנו לא דורשים שהחישוב מצליח בכל פעם, אלא מצליח בהסתברות גבוהה - בפרט החישוב יכול להיות שגוי לעיתים. בשנות השבעים הוצעו שני אלגוריתמים הסתברותיים לפתרון בעיה שעד אז לא היה ידוע עבורה אלגוריתם: האם מספר נתון הוא ראשוני? האלגוריתם הראשון היה של סולובייו ושטראסן [SS77], והשני של מילר ורבין [Rab80]. לא לקח זמן רב עד שאקראיות הפכה לכלי מרכזי בתכנון אלגוריתמים, ולתחום מחקר מרכזי בתורת הסיבוכיות. למרות זאת, במקרים רבים, וגם במקרה של בדיקת ראשוניות, נמצא כי בסופו של דבר האקראיות אינה הכרחית לחישוב יעיל. מתעוררת השאלה - באילו מקרים אקראיות אינה הכרחית לחישוב יעיל? האם ניתן לקחת אלגוריתם הסתברותי ולהפוך אותו לדטרמיניסטי מבלי להגדיל משמעותית את כמות המשאבים שזה צורך? התהליך של הסרת התלות במטבעות אקראיים, חלקית או לחלוטין, נקרא דה-רנדומיזציה. בהקשר של סיבוכיות זיכרון, הקהילה סבורה שעד כדי תקורה של כפל בקבוע, חישוב אקראי הינו שקול לחישוב קלאסי, קרי דטרמיניסטי.

אחת הדרכים המרכזיות לדה-רנדומיזציה בכלל, ובפרט של חישוב מוגבל זיכרון, היא באמצעות מחוללי פסאודו-אקראיות. בקצרה, מחולל פסאודו-אקראיות מייצר התפלגות שנראית אקראית עבור יריב מוגבל, במקרה שלנו אלגוריתמים מוגבלי זיכרון, ואיכותו נמדדת בשלושה פרמטרים:

א. אקראיות: כמות המטבעות האקראיים הנדרשת לייצר את ההתפלגות.

ב. זיכרון: כמות הזיכרון הנדרשת לייצר את התפלגות.

ג. שגיאה: מידת הדיוק שאלגוריתם מוגבל זיכרון יכול להבחין בין ההתפלגות הפסאודו-אקראית לבין ההתפלגות האחידה.

נרצה ששלושת הפרמטרים יהיו קטנים ככל האפשר. במאמר [CDR+21], שהוא עבודה משותפת עם גיל כהן, דין דורון, אמנון תא-שמע ואורן רנרד, אנחנו מראים כי בהינתן מחולל פסאודו-אקראיות עם שגיאה נתונה, ניתן להקטין את השגיאה עם תלות כמעט אופטימלית בזיכרון והאקראיות של המחולל. עם זאת, הבניה שלנו לא בונה מחולל פסאודו-אקראיות, אלא מחולל פסאודו-אקראיות ממושקל. תוצאה זו הושגה במקביל ובאופן בלתי תלוי על ידי טד פיין וסליל וודהאן [PV21]. הראשונים להבחין כי ניתן להוריד את השגיאה של מחוללי-פסאודו אקראיות על ידי בניה של מחולל ממושקל היו מארק ברוורמן, גיל כהן, וסומגה גארג [BCG20] שבנו מחולל ממושקל עם אקראיות

$$O(\log^2 n \cdot \log\log_n \frac{1}{\varepsilon} + \log n \cdot \log w + \log \frac{w}{\varepsilon} \cdot \log\log \frac{w}{\varepsilon}),$$

כאשר בעקבותיהם פורסמו בניות דומות [CL20, Hoz19]. לעבודה שלנו שני יתרונות על עבודות קודמות:

א. פשטות: הבניה פשוטה משמעותית מבניות קודמות [BCG20, CL20], והרעיונות מאחוריה מאד בסיסיים וקלים להבנה.

ב. כלליות: בעוד בניות קודמות תיארו מחולל ממושקל מסויים, הבניה שלנו מאפשר לקחת מחולל כלשהו עם פרמטרים כלשהו ולהוריד עבורו את השגיאה.

בנוסף, ייתכן מאד שניתן להחיל את הבניה, או ווריאנט שלה, למקרים נוספים.

התוצאה האחרונה מבוססת על עבודה משותפת עם גיל כהן, דין דורון, ואמנון תא-שמע [CDSTS22] ועוסקת בבעיה של דה-רנדומיזציה של אלגוריתמים הסתברותיים שזמן הריצה שלהם גדול משמעותית מהזיכרון שהם צורכים. בעבודה פורצת הדרך [SZ99] סאקס וז'ו הוכיחו כי כל אלגוריתם שרץ בזיכרון S ניתן לסמלץ באופן דטרמיניסטי באמצעות

$$O(S^{3/2})$$

זיכרון. התוצאה שלנו מתבססת על האלגוריתם שלהם, ומשיגה סימולציה שרצה בזיכרון משופר של $O(S\sqrt{\log T})$, כאשר T הוא זמן הריצה בתוחלת (אנחנו מרשים לאלגוריתם לא לעצור בהסתברות זניחה). השיפור בזיכרון בא

ניתן לבנות קודים מפורשים שמשיגים את הקיבול של ערוץ נתון. שאלה זו קיבלה מענה כאשר, בעבודה פורצת דרך, אריקן בנה קודים כאלה שלהם אלגוריתמי קידוד ופענוח יעילים באמצעות שיטת קיטוב האנטרופיה [Ar 09]. נעיר כי יש דקויות רבות במושג של השגת הקיבול, והיו תוצאות רבות גם עוד קודם, אך לא נרחיב על כך. למידע נוסף, ראו המאמר המקורי של אריקן.

מרבית העבודה על קודים לתיקון שגיאות במודל שאנון נעשתה באמצעות כלים מתורת האינפורמציה - אין זה פלא שכן עבודתו של שאנון נחשבת להולדת תורה זו. עם זאת, תורת הקודים מהר מאד התפתחה לכיוונים נוספים בהתאם לשינוי מודל הרעש, והדרישות מהקוד. האמינג הציע לתאר את הרעש על ידי יריב מוגבל [Ham50] ודרש כי תמיד ניתן לשחזר את מילת הקוד ללא תלות באסטרטגיה שהיריב בחר לייצר את הרעש - תמיד. באנאלוגיה לערוצי המחיקות/שגיאות שהציע שאנון, ליריב מותר למחוק/להפוך מספר מוגבל $t \in \mathbb{N}$ של קורדינטות במילת הקוד. באופן כמותי, אנחנו מצפים שדרושה יתירות גדולה בהרבה במודל של האמינג עבור כמות דומה של שגיאות שכן היריב בוחר בחוכמה אילו קורדינטות למחוק, בעוד שבמודל הרעש של שאנון אלו נבחרות אקראית. על כל פנים, במודל של האמינג הביצועים של הקוד נקבעים על פי סטטיסט בודד - המרחק בין שתי מילות הקוד הקרובות ביותר, כאשר קרבה נמדדת כמספר הקורדינטות השונות בערכן. פרמטר זה נקרא המרחק של הקוד. מודל האמינג הצית סוג של חדש של קודים שנבנו באמצעות בניות אלגבריות. דוגמאות חשובות כוללות את קודי ריד-מילר, קודי בוז-צ'הודרי-הוכנגהם, קודי גוליי, וקודים מאלגברה גיאומטרית. בדרך כלל, קודים אלו מקיימים תכונות מבניות שימושיות כגון טרנזיטיביות, טרנזיטיביות-כפולה, מעגליות, דואליות ועוד, שמנוצלות לצורך שימושים שונים בתורת הקודים, מה שהופך אותם למושכים במיוחד.

למרות שהביצועים של קודים אלו במודל האמינג ידועים במידה רבה, הניתוח שלהם במודל שאנון נשאר בגדר תעלומה. לפחות עבור קודי ריד-מילר, הקהילה שיערה כי אלו משיגים את הקיבול של כל ערוץ נתון, כלומר הביצועים שלהם אופטימליים במודל של שאנון. למעשה, ניסויים אמפיריים הצביעו על כך שהביצועים שלהם טובים יותר מהקודים של אריקן [AKM+09]. רק בשנת 2015 הצליחו קודקר, קומר, מונדלי, פיסטר, ססוגלו, ואורבנקה להוכיח כי קודי ריד-מילר עם קצב קבוע משיגים את הקיבול של ערוץ המחיקות [KKM+17]. באותה שנה, אבה, שפילקה, ו-ויגדרזון הראו כי גם קודי ריד-מילר עם קצב שואף לאפס מספיק מהר, וקצב שואף לאחד מספיק מהר, משיגים את הקיבול של ערוץ המחיקות והשגיאות [ASW15, SS20].

התוצאה המרכזית שלנו היא שבמובן מסויים אם הקוד יכול להתמודד עם רעש בערוץ כלשהו, אז הוא יכול להתמודד עם רעש בכל ערוץ. בהתבסס על העבודה של קודקר ושותפיו [KKM+17] על הביצועים של קודים המקיימים את תכונות הטרנזיטיביות הכפולה מעל ערוץ מחיקות, שזהו ערוץ מסויים במודל שאנון, אנחנו מסיקים את התוצאה הבאה: כל קוד כנ"ל יכול להתמודד עם רעש מעל כל ערוץ שאינו רועש דיו. הניסוח המדויק דורש מושג שנקרא פרמטר בטצ'ריה, שמודד את מידת הרעש בערוץ. במונחים של הערוצים שראינו ניתן לנסח את התוצאה באופן הבא: כל קוד שמקיים את תכונת הטרנזיטיביות הכפולה אמין מעל ערוץ המחיקות עם פרמטר p_1, מעל ערוץ השגיאות הסימטרי עם פרמטר p_2, ואמין מעל הערוץ הגאוסי עם שונות σ כאשר p_1, p_2, σ קבועים התלויים בקצב של הקוד, ובמרחק של הקוד.

העבודה של קודקר ושותפיו מניחה כי המרחק של הקוד גבוה למדי, אך אנחנו מצליחים להחליש הנחה זו בצורה משמעותית. מסקנה חשובה נוספת מהעבודה שלנו היא שקודי ריד-מילר, מקודי השגיאות המוכרים והחשובים בתחום, יכולים להתמודד עם רעש מעל כל ערוץ. נעיר כי במאמר שפורסם לאחר מכן [RP21] הוכח כי למעשה קודי ריד-מילר יכולים להתמודד עם כמות רעש ששואפת לקיבולת של כל ערוץ. למען הדיוק, התוצאה שלהם מדברת על פענוח בתוחלת, כלומר שבתוחלת משחזרים **כמעט** את כל מילת הקוד, להבדיל משחזור **מלא** של מילת הקוד בהסתברות גבוהה.

התוצאות מבוסס על עבודה משותפת עם יאן הזלה, ואלכס סמרודניצקי [HSS21].

פרק ב' - חישוב אקראי מוגבל זיכרון

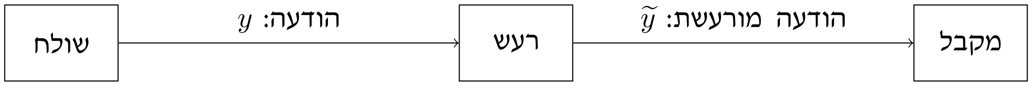
תורת הסיבוכיות עוסקת בכמות המשאבים הנדרשת לפתרון בעיה נתונה כגון זמן ריצה, זיכרון, זמן ריצה מקבילי, תקשורת, אקראיות, ועוד. בתיאוריה של מדעי המחשב, נהוג לתאר חישוב באמצעות המודל של טיורינג - מכונת טיורינג (מ"ט). בקצרה, מ"ט היא סדרה סופית של הוראות (אלגוריתם) הפועלת על סרט אינסופי, לרוב חד-כיווני, שממנו היא יכולה לקרוא ולכתוב.

בעבודה אנחנו מתרכזים במשאבי הזיכרון והאקראיות אותם נסביר כעת. במונחים של מ"ט, הזיכרון הוא אורך הסרט שהמכונה משתמשת בו בפועל, כאשר אנחנו מתעלמים מהמקום שדרוש לשמור את הקלט לבעיה, והפלט.

תקציר

פרק א' - קודים לתיקון שגיאות במודל שאנון

בעבודתו פורצת הדרך "תאוריה מתמטית של התקשורת" [Sha48] קלוד שאנון התעניין בשאלה של תקשורת בין שני צדדים מעל תווך רועש



עבודתו הניחה את היסודות לתורת הקודים, שלימים הפכה לאחד התחומים החשובים ביותר במדעי המחשב, ונמצאה שימושית הרבה מעבר לבעיה המקורית של תקשורת מעל תווך רועש.

בעבודה אנחנו מתמקדים בקודים בינארים, לינארים, לתיקון שגיאות, ובביצועים שלהם במודל הרעש שהוצע על ידי שאנון. קוד בינארי, לינארי לתיקון שגיאות הינו תת-מרחב לינארי $V \subseteq \mathbb{F}_2^n$, והווקטורים במרחב נקראים גם מילות קוד. ניתן לזהות את המרחב עם העתקה לינארית

$$E\colon \mathbb{F}_2^{\dim(V)} \to \mathbb{F}_2^n$$

שנקראת גם פונקצית הקידוד, אך נהוג להתעלם מזו ולהתרכז במרחב V. כאמור, V צריך לקיים את התכונה שבהינתן גרסה מורעשת של ווקטור כלשהו $v \in V$, ניתן לחלץ מידע על v וככל שניתן גם לשחזרו. הדבר מתאפשר כי אורכו של v הינו n, בעוד הוא שייך למרחב ממימד קטן יותר קרי

$$\dim(V) < n,$$

או במילים אחרות, קיימת יתירות בייצוג של v. כמות היתירות נמדדת באמצעות פרמטר שנקרא קצב ומוגדר על ידי

$$R(V) \overset{\text{def}}{=} \frac{\dim(V)}{n}.$$

הקצב, הינו אחד הפרמטרים הבסיסיים ביותר של קוד לתיקון שגיאות. נעיר כי יתירות אינה ערובה ליכולת לתיקון שגיאות, כלומר עצם קיומה של יתירות אינו מבטיח את היכולת לשחזר את המידע הנדרש על v - כאן נכנסת תורת הקודים, והמטרה היא לבחור את המרחב V בחוכמה, ובהתאם לדרישות מהקוד.

כדי לעסוק באופן מתמטי בשאלה של תיקון שגיאות, יש להגדיר כיצד שגיאות אלו נוצרות. בעבודותו, שאנון הציע לתאר את הרעש כתהליך אקראי, חסר זיכרון, ושאל האם ניתן לשחזר את מילת הקוד בהסתברות גבוהה. להלן שלושה תהליכים אקראיים, הנקראים גם ערוצים, שמהווים מודלים נפוצים לתיאור רעש:

א. ערוץ מחיקות: כל קורדינטה מוחלפת בסימן שאלה '?' בהסתברות $p \in (0,1)$, באופן בלתי תלוי.

ב. ערוץ שגיאות סימטרי: כל קורדינטה 'מתהפכת', קרי '0' הפוך ל- '1' ולהיפך, בהסתברות $p \in (0, \frac{1}{2})$ באופן בלתי תלוי.

ג. רעש לבן גאוסיאני: לכל קורדינטה מתווסף רעש המתפלג בצורה גאוסיאנית, עם שונות נתונה σ. בערוץ זה נהוג לפרש את הקורדינטות בתור $\{-1,1\}$ במקום $\{0,1\}$.

נעיר כי למעשה מדובר בשלוש משפחות של ערוצים, שכן כל בחירה של פרמטר הרעש p במקרה של מחיקות/שגיאות, או σ במקרה של רעש לבן גאוסיאני, מגדירה ערוץ. שלושת הערוצים לעיל שייכים למשפחה של ערוצים הנקראים ערוצים-סימטרים-חסרי-זיכרון, ומעתה כאשר נאמר ערוץ, אנחנו מתכוונים לערוץ כזה. עבור ערוץ וקוד נתונים, נאמר כי הקוד אמין מעל הערוץ אם ניתן לשחזר בהסתברות גבוהה מילות קוד שהורעשו על ידיו.

בעבודתו, שאנון הוכיח כי לכל ערוץ יש חסם עליון על קצב הקוד שניתן באמצעותו לתקשר באמינות מעל הערוץ. קצב מרבי זה נקרא הקיבול של הערוץ. לדוגמה, לא ניתן לשדר באמינות קוד עם קצב גדול מ-$(1-p)$ מעל ערוץ מחיקות עם הסתברות מחיקה p, כלומר קיבול ערוץ המחיקות הוא $1-p$. יתרה מזאת, הוא הוכיח כי לכל ערוץ קיימים קודים שמשיגים חסם זה, אך ההוכחה שלו לא בונה קוד מפורש שכזה. נותרה השאלה, האם וכיצד

תמצית

החלק הראשון של התזה עוסק בקודים לתיקון שגיאות תחת מודל הרעש של שאנון ובו נראה כי כל קוד שיכול להתמודד עם רעש בערוץ כלשהו, יכול להתמודד עם רעש מעל כל ערוץ. בהתבסס על עבודתם של קודקר, קומר, מונדלי, פיסטר, ססוגלו, ואורבנקה [KKM$^+$17] אנחנו מסיקים את התוצאה הבאה: כל קוד שמקיים את תכונת הטרנזיטיביות הכפולה יכול להתמודד עם רעש מעל כל ערוץ עם קבוע בטצ'ריה מספיק קטן. בעוד שהעבודה של קודקר ושותפיו מניחה כי המרחק של הקוד גבוה למדי, אנחנו מצליחים להחליש הנחה זו בצורה משמעותית. מסקנה נוספת מהעבודה שלנו היא שקודי ריד-מילר, מקודי השגיאות המוכרים והחשובים בתחום, יכולים להתמודד עם רעש מעל כל ערוץ. נעיר כי במאמר שפורסם לאחר מכן [RP21] הוכח כי למעשה קודי ריד-מילר יכולים להתמודד עם כמות רעש ששואפת לקיבול של כל ערוץ, או במילים אחרות, מתמודדים עם הכמות המירבית של רעש שניתן להתמודד עמה.

בחלק השני של התזה נדון בדה-רנדומיזציה של חישוב מוגבל זיכרון. תחילה, נתמקד במחוללי פסאודו-אקראיות ונוכיח כי בהינתן מחולל פסאודו-אקראיות עם שגיאה נתונה, ניתן להקטין את השגיאה עם תלות כמעט אופטימלית בזיכרון והאקראיות של המחולל. עם זאת, הבניה שלנו לא בונה מחולל פסאודו-אקראיות, אלא מחולל פסאודו-אקראיות ממושקל. תוצאה זו הושגה במקביל ובאופן בלתי תלוי על ידי טד פיין וסליל וודהאן [PV21].

לאחר מכן נדון בבעיה של קירוב חזקות של מטריצות סטוכסטיות, שבמובן מסוים תופסת את הבעיה של דה-רנדומיזציה של אלגוריתמים מוגבלי זיכרון. אם נתעלם לרגע מדקויות, קירוב של חזקות מסדר T למטריצות סטוכסטיות ממימד $2^S \times 2^S$ הינו שקול לדה-רנדומיזציה של אלגוריתמים הסתברותיים שרצים בזמן T וזיכרון S. סאקס וז'ו תיארו אלגוריתם [SZ99] לבעיה של קירוב חזקות של מטריצות סטוכסטיות שרץ בזיכרון

$$O((S + \log T)\sqrt{\log T}).$$

התוצאה שלנו מתבססת על האלגוריתם שלהם, ומשיגה סימולציה שרצה בזיכרון משופר של $O(S\sqrt{\log T})$. השיפור בא לידי ביטוי כאשר $S \ll \log T$. לכן, כאשר מתרגמים את התוצאה בחזרה לעולם של דה-רנדומיזציה, השיפור תקף לאלגוריתמים שבהם זמן הריצה גדול משמעותית מהזיכרון שהם צורכים. אלגוריתמים כאלה בהכרח אינם עוצרים (לעיתים), ולכן חשוב להדגיש כי אנחנו מאפשרים לאלגוריתם לא לעצור בהסתברות זניחה ודורשים כי זמן הריצה הוא בתוחלת T. מה לגבי קירוב מכפלה של מטריצות סטוכסטיות נתונות

$$A_1, \ldots, A_n \in \mathbb{R}^{w \times w}?$$

האלגוריתם היעיל ביותר שהיה ידוע לבעיה היה האלגוריתם לעיל שכאמור רץ בזיכרון $O((\log w + \log n)\sqrt{\log n})$. נתאר אלגוריתם לבעיה זו שרץ בערך בזיכרון

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

כלומר משפר את סיבוכיות הזיכרון בתחום $w \ll n$. כדי להדגים את ההבדל בפרמטרים נתבונן במקרה שבו

$$w = 2^{\sqrt{\log n}},$$

קרי יש לנו n מטריצות ממימד $2^{\sqrt{\log n}}$. האלגוריתם של סאקס וז'ו רץ בזיכרון של $O(\log^{3/2} n)$ עבור פרמטרים אלו, ולמעשה ניתן להשיג אלגוריתם עם פרמטרים זהים בקלות ללא שימוש באלגוריתם של סאקס וז'ו. לעומת זאת, האלגוריתם שלנו משיג סיבוכיות זיכרון כמעט מיטבית של $O(\log n \log\log n)$.

החלק הראשון של התזה מבוסס על עבודה משותפת עם יאן הזלה, ואלכס סמורודניצקי [HSS21], והחלק השני על עבודות בשיתוף גיל כהן, דין דורון, אורן רנרד, ואמנון תא-שמע [CDR$^+$21, CDSTS22].

אוניברסיטת תל אביב
הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר
ביה"ס למדעי המחשב ע"ש בלווטניק

קודים לתיקון שגיאות ודה-רנדומיזציה של חישוב מוגבל זיכרון

חיבור לשם קבלת התואר
דוקטור לפילוסופיה
מאת
אורי סברלו

מנחים: פרופ' גיל כהן, פרופ' אמנון תא-שמע.

הוגש לסנאט של אוניברסיטת תל אביב
ספטמבר 2022