Randomized Methods in Computation:

Tentative Collection of Reading Material

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

July 16, 2011

Preface

I was advised to start the preface with the following couple of preliminary remarks. An overview of the material that is presented in the body of this text follows.

Two Preliminary Remarks

By *Randomized Methods in Computation* I mean the collection of tools, methods, and perceptions that have an explicit or implicit *randomized flavour* and are *used in the analysis of computation*. Typically, but not always, the computation being analyzed is randomized, and in such cases the use of probabilistic analysis is explicit. Nevertheless, this text is not a course in probability theory; material that can be labeled as classical "probability theory" occupies only the first five (or so) pages of the first chapter (and material that can be labeled "probability theory" appears only in the first chapter).[1]

This tentative collection of reading material regarding *Randomized Methods in Computation* was compiled based on extracts taken from a variety of sources. Some of the "chapters" are extracts from my book on *computational complexity*, while others reproduce surveys that I wrote on different occasions and for different audiences. Consequently, this collection suffers from some drawbacks (which also offer some advantages). Most importantly:

- Some material appears in more than one of the "chapters" (but the perspectives offered are usually different). Furthermore, in some cases there are inconsistencies (between "chapters") regarding the exact formulations and notation.
- Some "chapters" put a greater emphasis on the results obtained while others put more emphasis on the techniques used. (This may be OK; forcing uniformity here may be undesired.)
- The expositional style varies from a novice-friendly style of a textbook to a laconic style aimed at experts. (Even this may have a bright side: It exposes the novice to the style that prevails in the literature.)
- Lastly, this text still contains some typos and minor errors.

Hopefully, I will revise this collection in the future and eliminate some of its drawbacks, but for the time being this collection is what I have to offer.

[1] Indeed, the second part of the first chapter contains material that may be labeled "probability theory", but I doubt one can find this material in any standard textbook on probability theory.

Overview

A variety of randomized methods are being employed in the study of computation. The aim of the current course is to make the students familiar with some of these methods. We wish to stress three aspects regarding this course:

1. This course focuses on *methods* (i.e., tools and techniques) and not on concepts.
2. These methods are closely related to *randomization*.
3. The focus is on *applications to the study of computation*.

Specific topics included:

- Elements of Probability Theory: Linearity of expectation, Markov Inequality, Chebyshev's Inequality, and Laws of Large Numbers (for pairwise-independent, k-wise independent, and fully independent samples (aka Chernoff Bound)).
- The *Probabilistic Method*: Applications to the existence of good error-correcting codes and 3-regular expander graphs.
- Pairwise-independence, hashing, and related topics: Constructions (via matrices and low-degree polynomials), and applications (to Approximate Counting and Uniform Generation).
- Small bias spaces: constructions and applications.
- Expander graphs and random walks on them: Eigenvalues versus expansion, analysis of random walks (Hitter), and the Median-of-Average Sampler.
- Randomness-Extractors: brief overview of constructions and applications.
- Some randomized algorithms: Undirected Connectivity (via a random walk), Polynomial Identity Testing, and Primality Testing (via SQRT extraction).

Recurring notation: For any natural number n, we denote $[n] \stackrel{\text{def}}{=} \{1, 2, ..., n\}$.

Additional sources

Lecture notes of my 2001 course on the subject are available from

`http://www.wisdom.weizmann.ac.il/~oded/rnd.html`.

In addition, there are several books that cover parts of the material. These include:

- N. Alon and J.H. Spencer: *The Probabilistic Method*, John Wiley & Sons, Inc., 1992.
- O. Goldreich: *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, 2008. Drafts are available from `http://www.wisdom.weizmann.ac.il/~oded/cc-drafts.html`.
- R. Motwani and P. Raghavan: *Randomized Algorithms*, Cambridge University Press, 1995.

Additional related material can be found in

- D.P. Dubhashi and A. Panconesi: *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.
- M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005

Acknowledgments

I am grateful to Omer Pinto for encouraging me to compile and revise these notes, and for leading a group of Weizmann students that read an early version of this document in Spring 2011. This group consisting of Alina Arbitman, Tomer Ashur, Gilad Braunschvig, Tom Gur, Nimrod Talmon, and Eldad Zinger, provided me with helpful feedback of various types.

Contents

Lecture 1

Probabilistic Preliminaries

The probabilistic preliminaries include our conventions regarding random variables, which are used throughout the book. In addition we provide overviews of three useful inequalities: Markov Inequality, Chebyshev's Inequality, and Chernoff Bound. Also included are two (purely probabilistic) XOR Lemmas.

1.1 Notational Conventions

We assume that the reader is familiar with the basic notions of probability theory. In this section, we merely present the probabilistic notations that are used throughout the text.

Throughout the entire course we will refer only to *discrete* probability distributions. Specifically, the underlying probability space will consist of the set of all strings of a certain length ℓ, taken with uniform probability distribution. That is, the sample space is the set of all ℓ-bit long strings, and each such string is assigned probability measure $2^{-\ell}$. Traditionally, *random variables* are defined as functions from the sample space to the reals. Abusing the traditional terminology, we use the term random variable also when referring to functions mapping the sample space into the set of binary strings. We often do not specify the probability space, but rather talk directly about random variables. For example, we may say that X is a random variable assigned values in the set of all strings such that $\mathsf{Pr}[X=00] = \frac{1}{4}$ and $\mathsf{Pr}[X = 111] = \frac{3}{4}$. (Such a random variable may be defined over the sample space $\{0,1\}^2$, so that $X(11) = 00$ and $X(00) = X(01) = X(10) = 111$.) One important case of a random variable is the output of a randomized process (e.g., a probabilistic polynomial-time algorithm).

All our probabilistic statements refer to functions of random variables that are defined beforehand. Typically, we may write $\mathsf{Pr}[f(X)=1]$, where X is a random variable defined beforehand (and f is a function). An important convention is that *all occurrences of the same symbol in a probabilistic statement refer to the same* (unique) *random variable*. Hence, if $B(\cdot,\cdot)$ is a Boolean expression depending on two variables, and X is a random variable then $\mathsf{Pr}[B(X,X)]$ denotes the probability that $B(x,x)$ holds when x is chosen with probability $\mathsf{Pr}[X = x]$. For example, for every random variable X, we have $\mathsf{Pr}[X = X] = 1$. We stress that if we wish to discuss the probability that $B(x,y)$ holds when x and y are chosen

independently with identical probability distribution, then we will define *two* independent random variables each with the same probability distribution. Hence, if X and Y are two independent random variables then $\mathsf{Pr}[B(X,Y)]$ denotes the probability that $B(x,y)$ holds when the pair (x,y) is chosen with probability $\mathsf{Pr}[X=x] \cdot \mathsf{Pr}[Y=y]$. For example, for every two independent random variables, X and Y, we have $\mathsf{Pr}[X=Y] = 1$ only if both X and Y are trivial (i.e., assign the entire probability mass to a single string).

Throughout the entire book, U_n denotes a random variable uniformly distributed over the set of strings of length n. Namely, $\mathsf{Pr}[U_n = \alpha]$ equals 2^{-n} if $\alpha \in \{0,1\}^n$ and equals 0 otherwise. We will often refer to the distribution of U_n as the uniform distribution (neglecting to qualify that it is uniform over $\{0,1\}^n$). In addition, we will occasionally use random variables (arbitrarily) distributed over $\{0,1\}^n$ or $\{0,1\}^{\ell(n)}$, for some function $\ell : \mathbb{N} \to \mathbb{N}$. Such random variables are typically denoted by X_n, Y_n, Z_n, etc. We stress that in some cases X_n is distributed over $\{0,1\}^n$, whereas in other cases it is distributed over $\{0,1\}^{\ell(n)}$, for some function $\ell(\cdot)$, which is typically a polynomial. We will often talk about probability ensembles, which are infinite sequence of random variables $\{X_n\}_{n\in\mathbb{N}}$ such that each X_n ranges over strings of length bounded by a polynomial in n.

Statistical difference. The statistical distance (a.k.a variation distance) between the random variables X and Y is defined as

$$\frac{1}{2} \cdot \sum_v |\mathsf{Pr}[X=v] - \mathsf{Pr}[Y=v]| \;=\; \max_S \{\mathsf{Pr}[X \in S] - \mathsf{Pr}[Y \in S]\}. \tag{1.1}$$

(Showing the equality is deferred to Exercise 1.1.) We say that X is δ-close (resp., δ-far) to Y if the statistical distance between them is at least (resp., at most) δ.

Expectation and variance. Recall that the expectation of a random variable X, denoted $\mathsf{E}(X)$, is defined as $\sum_v \mathsf{Pr}[X=v] \cdot v$. The variance of X, denoted $\mathsf{Var}(X)$, is defined as $\mathsf{E}((X - \mathsf{E}(X))^2)$. Proving the following properties of the expectation operator is left for Exercise 1.2:

- *Linearity of expectation*: For every two random variables (not necessarily independent ones), X and Y, it holds that $\mathsf{E}(X+Y) = \mathsf{E}(X) + \mathsf{E}(Y)$.
- *Alternative form for variance*: $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$.

1.2 Three Inequalities

The following probabilistic inequalities are very useful. These inequalities refer to random variables that are assigned real values and provide upper-bounds on the probability that the random variable deviates from its expectation.

1.2.1 Markov's Inequality

The most basic inequality is Markov's Inequality that applies to any random variable with bounded maximum or minimum value. For simplicity, this inequality is stated for random

variables that are lower-bounded by zero, and reads as follows: *Let X be a non-negative random variable and v be a non-negative real number. Then*

$$\mathsf{Pr}\left[X \geq v\right] \leq \frac{\mathsf{E}(X)}{v} \tag{1.2}$$

Equivalently, $\mathsf{Pr}[X \geq r \cdot \mathsf{E}(X)] \leq \frac{1}{r}$. The proof amounts to the following sequence:

$$\begin{aligned} \mathsf{E}(X) &= \sum_x \mathsf{Pr}[X=x] \cdot x \\ &\geq \sum_{x<v} \mathsf{Pr}[X=x] \cdot 0 + \sum_{x \geq v} \mathsf{Pr}[X=x] \cdot v \\ &= \mathsf{Pr}[X \geq v] \cdot v \end{aligned}$$

1.2.2 Chebyshev's Inequality

Using Markov's inequality, one gets a potentially stronger bound on the deviation of a random variable from its expectation. This bound, called Chebyshev's inequality, is useful when having additional information concerning the random variable (specifically, a good upper bound on its variance). For a random variable X of finite expectation, we denote by $\mathsf{Var}(X) \stackrel{\text{def}}{=} \mathsf{E}[(X - \mathsf{E}(X))^2]$ the variance of X, and observe that $\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$. Chebyshev's Inequality then reads as follows: *Let X be a random variable, and $\delta > 0$. Then*

$$\mathsf{Pr}\left[|X - \mathsf{E}(X)| \geq \delta\right] \;\leq\; \frac{\mathsf{Var}(X)}{\delta^2} \;. \tag{1.3}$$

Proof: We define a random variable $Y \stackrel{\text{def}}{=} (X - \mathsf{E}(X))^2$, and apply Markov's inequality. We get

$$\begin{aligned} \mathsf{Pr}\left[|X - \mathsf{E}(X)| \geq \delta\right] &= \mathsf{Pr}\left[(X - \mathsf{E}(X))^2 \geq \delta^2\right] \\ &\leq \frac{\mathsf{E}[(X - \mathsf{E}(X))^2]}{\delta^2} \end{aligned}$$

and the claim follows. ■

Corollary (Pairwise Independent Sampling)**:** Chebyshev's inequality is particularly useful in the analysis of the error probability of approximation via repeated sampling. It suffices to assume that the samples are picked in a pairwise independent manner, where $X_1, X_2, ..., X_n$ are pairwise independent if for every $i \neq j$ and every α, β it holds that $\mathsf{Pr}[X_i = \alpha \wedge X_j = \beta] = \mathsf{Pr}[X_i = \alpha] \cdot \mathsf{Pr}[X_j = \beta]$. The corollary reads as follows: *Let $X_1, X_2, ..., X_n$ be pairwise independent random variables with identical expectation, denoted μ, and identical variance, denoted σ^2. Then, for every $\varepsilon > 0$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] \leq \frac{\sigma^2}{\varepsilon^2 n} \;. \tag{1.4}$$

Proof: Define the random variables $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$. Note that the $\overline{X}_i$'s are pairwise independent, and each has zero expectation. Applying Chebyshev's inequality to the random variable $\sum_{i=1}^n \frac{X_i}{n}$, and using the linearity of the expectation operator, we get

$$\begin{aligned}\mathsf{Pr}\left[\left|\sum_{i=1}^n \frac{X_i}{n} - \mu\right| \geq \varepsilon\right] &\leq \frac{\mathsf{Var}\left[\sum_{i=1}^n \frac{X_i}{n}\right]}{\varepsilon^2} \\ &= \frac{\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right]}{\varepsilon^2 \cdot n^2}\end{aligned}$$

Now (again using the linearity of expectation)

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = \sum_{i=1}^n \mathsf{E}\left[\overline{X}_i^2\right] + \sum_{1\leq i\neq j\leq n} \mathsf{E}\left[\overline{X}_i \overline{X}_j\right]$$

By the pairwise independence of the $\overline{X}_i$'s, we get $\mathsf{E}[\overline{X}_i \overline{X}_j] = \mathsf{E}[\overline{X}_i] \cdot \mathsf{E}[\overline{X}_j]$, and using $\mathsf{E}[\overline{X}_i] = 0$, we get

$$\mathsf{E}\left[\left(\sum_{i=1}^n \overline{X}_i\right)^2\right] = n \cdot \sigma^2$$

The corollary follows. ■

Generalization: k-Wise Independent Sampling. The foregoing corollary can be generalized to random variables with higher level of independence. Specifically, for $k \in \{2, .., n\}$, we say that $X_1, X_2, ..., X_n$ are k-wise independent if for every $i_1 < i_2 < \cdots < i_k$ and every $\alpha_1, \alpha_2, ..., \alpha_k$ it holds that $\mathsf{Pr}[\wedge_{j=1}^k X_{i_j} = \alpha_j]$ equals $\prod_{j=1}^k \mathsf{Pr}[X_{i_j} = \alpha_j]$. The generalized corollary reads: *Let $X_1, X_2, ..., X_n$ be k-wise independent random variables with identical expectation, denoted μ, and variance at most 1. Then, for every $\varepsilon > 0$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right] < \left(\frac{2k}{\varepsilon^2 n}\right)^{\lfloor k/2 \rfloor} \tag{1.5}$$

The proof is implicit in the proof of Lemma 4.6; see also Exercise 1.6. Note that Eq. (1.5) is a simplified form, which does not imply the previous corollary as a special case (since it assumes $\sigma^2 \leq 1$ and looses a factor of $2k/\sigma^2$).

1.2.3 Chernoff Bound

When using pairwise independent sample points, the error probability in the approximation decreases linearly with the number of sample points (see Eq. (1.4)). When using totally independent sample points, the error probability in the approximation can be shown to decrease exponentially with the number of sample points. (Recall that the random variables $X_1, X_2, ..., X_n$ are said to be totally independent if for every sequence $a_1, a_2, ..., a_n$ it holds that $\mathsf{Pr}[\wedge_{i=1}^n X_i = a_i] = \prod_{i=1}^n \mathsf{Pr}[X_i = a_i]$.) Probability bounds supporting the foregoing

statement are given next. The first bound, commonly referred to as Chernoff Bound, concerns 0-1 random variables (i.e., random variables that are assigned as values either 0 or 1), and asserts the following. *Let $p \leq \frac{1}{2}$, and $X_1, X_2, ..., X_n$ be independent 0-1 random variables such that $\mathsf{Pr}[X_i = 1] = p$, for each i. Then, for every $\varepsilon \in (0, p]$, it holds that*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - p\right| > \varepsilon\right] \;<\; 2 \cdot e^{-c \cdot \varepsilon^2 \cdot n}\,, \text{ where } c = \max(2, \tfrac{1}{3p}). \tag{1.6}$$

The more common formulation sets $c = 2$, but the case $c = 1/3p$ is very useful when p is small and one cares about a *multiplicative* deviation (e.g., $\varepsilon = p/2$).

Proof Sketch: We upper-bound $\mathsf{Pr}[\sum_{i=1}^{n} X_i - pn > \varepsilon n]$, and $\mathsf{Pr}[pn - \sum_{i=1}^{n} X_i > \varepsilon n]$ is bounded similarly. Letting $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$, we apply Markov's inequality to the random variable $e^{\lambda \sum_{i=1}^{n} \overline{X}_i}$, where $\lambda \in (0, 1]$ will be determined to optimize the expressions that we derive. Thus, $\mathsf{Pr}[\sum_{i=1}^{n} \overline{X}_i > \varepsilon n]$ is upper-bounded by

$$\frac{\mathsf{E}[e^{\lambda \sum_{i=1}^{n} \overline{X}_i}]}{e^{\lambda \varepsilon n}} = e^{-\lambda \varepsilon n} \cdot \prod_{i=1}^{n} \mathsf{E}[e^{\lambda \overline{X}_i}]$$

where the equality is due to the independence of the random variables. To simplify the rest of the proof, we establish a sub-optimal bound as follows. Using a Taylor expansion of e^x (e.g., $e^x < 1 + x + x^2$ for $|x| \leq 1$) and observing that $\mathsf{E}[\overline{X}_i] = 0$, we get $\mathsf{E}[e^{\lambda \overline{X}_i}] < 1 + \lambda^2 \mathsf{E}[\overline{X}_i^2]$, which equals $1 + \lambda^2 p(1-p)$. Thus, $\mathsf{Pr}[\sum_{i=1}^{n} X_i - pn > \varepsilon n]$ is upper-bounded by $e^{-\lambda \varepsilon n} \cdot (1 + \lambda^2 p(1-p))^n < \exp(-\lambda \varepsilon n + \lambda^2 p(1-p) n)$, which is optimized at $\lambda = \varepsilon/(2p(1-p))$ yielding $\exp(-\frac{\varepsilon^2}{4p(1-p)} \cdot n) \leq \exp(-\varepsilon^2 \cdot n)$. □

The foregoing proof strategy can be applied in more general settings.[1] A more general bound, which refers to independent random variables that are each bounded but are not necessarily identical, is given next (and is commonly referred to as Hoefding Inequality). *Let $X_1, X_2, ..., X_n$ be n independent random variables, each ranging in the (real) interval $[a, b]$, and let $\mu \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \mathsf{E}(X_i)$ denote the average expected value of these variables. Then, for every $\varepsilon > 0$,*

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - \mu\right| > \varepsilon\right] < 2 \cdot e^{-\frac{2\varepsilon^2}{(b-a)^2} \cdot n} \tag{1.7}$$

The special case (of Eq. (1.7)) that refers to identically distributed random variables is easy to derive from the foregoing Chernoff Bound (by recalling Footnote 1 and using a linear mapping of the interval $[a, b]$ to the interval $[0, 1]$). This special case is useful in estimating the average value of a (bounded) function defined over a large domain, especially when the desired error probability needs to be negligible (i.e., decrease faster than any polynomial in the number of samples). Such an estimate can be obtained provided that we can sample the function's domain (and evaluate the function).

[1] For example, verify that the current proof actually applies to the case that $X_i \in [0, 1]$ rather than $X_i \in \{0, 1\}$, by noting that $\mathsf{Var}[X_i] \leq p(1-p)$ still holds.

1.2.4 Pairwise independent versus totally independent sampling

To demonstrate the difference between the sampling bounds provided in Section 1.2.2 and Section 1.2.3, we consider the problem of estimating the average value of a function $f : \Omega \to [0,1]$. In general, we say that a random variable Z provides an (ε,δ)-approximation of a value v if $\mathsf{Pr}[|Z - v| > \varepsilon] \leq \delta$. By Eq. (1.7), the average value of f evaluated at $n = O((\varepsilon^{-2} \cdot \log(1/\delta))$ *independent* samples (selected uniformly in Ω) yield an (ε,δ)-approximation of $\mu = \sum_{x\in\Omega} f(x)/|\Omega|$. Thus, the number of sample points is polynomially related to ε^{-1} and logarithmically related to δ^{-1}. In contrast, by Eq. (1.4), an (ε,δ)-approximation by n *pairwise independent* samples calls for setting $n = O(\varepsilon^{-2} \cdot \delta^{-1})$. We stress that, *in both cases the number of samples is polynomially related to the desired accuracy of the estimation* (i.e., ε). *The only advantage of totally independent samples over pairwise independent ones is in the dependency of the number of samples on the error probability* (i.e., δ).

1.3 Two Information Theoretic XOR Lemmas

We present two (information theoretic) XOR lemmas, which refer to 0-1 random variables and to the effect of XORing certain subsets of these variables. The first lemma refers to *totally independent* but possibly biased random variables, and asserts that *when XORing all variables the bias of the result* (i.e., its deviation from a uniformly distributed random bit) *vanishes exponentially with the number of variables that are being XORed*. The second lemma refers to an arbitrary sequence of (potentially dependent) random variables, and *relates the deviation of their joint distribution from the uniform distribution to the biases of the XOR of all subsets of variables*.

1.3.1 The bias of the XOR of all variables

The following lemma is presented for three reasons. Firstly, it is quite a useful fact, which is often used as is. Secondly, it is a simple analogue of the celebrated XOR Lemma of Yao (which refers to computational unpredictability, see [47] or [44, §7.2.1.2]). Thirdly, its proof reveals the great benefit that can be obtained by a simple change in notation: Moving from $\{0,1\}$ to $\{\pm 1\}$ by $x \mapsto (-1)^x$.

Lemma 1.1 (XORing totally independent 0-1 random variables): Let $p \in [0,1]$, and $X_1, X_2, ..., X_n$ be totally independent 0-1 random variables such that $\mathsf{Pr}[X_i = 1] = p$, for each i. Then, $\mathsf{Pr}[\sum_{i=1}^n X_i \equiv 1 \pmod 2] = \frac{1}{2} - \frac{(1-2p)^n}{2}$.

Proof: We start by moving to ± 1 notation:

$$\mathsf{Pr}\left[\sum_{i=1}^{n} X_i \equiv 0 \pmod 2\right] = \mathsf{Pr}\left[\prod_{i=1}^{n} (-1)^{X_i} = 1\right] \tag{1.8}$$

Next, we use the fact that for a random variable Z obtaining values in $\{\pm 1\}$, it holds that

$$\mathsf{E}[Z] \;=\; \mathsf{Pr}\left[Z = 1\right] - \mathsf{Pr}\left[Z = -1\right] \;=\; 1 - 2 \cdot \mathsf{Pr}\left[Z = -1\right]. \tag{1.9}$$

Combining Eq. (1.8) and Eq. (1.9), the lemma's claim is restated as

$$\mathsf{E}\left[\prod_{i=1}^{n}(-1)^{X_i}\right] = (1-2p)^n. \tag{1.10}$$

Using the hypothesis that the X_i's are totally independent (and $\mathsf{E}[(-1)^{X_i}] = 1-2p$), Eq. (1.10) follows. (Indeed, if $X_1, ..., X_n$ are independent, then $f(X_1), ..., f(X_n)$ are independent for any function f; see Exercise 1.5.) ■

Digest. Indeed, the punchline of the forgoing proof is that for totally independent random variables, $Z_1, ..., Z_n$, in $\mathbb{R}$ it holds that $\mathsf{E}[\prod_{i=1}^{n} Z_i] = \prod_{i=1}^{n} \mathsf{E}[Z_i]$. This fact can be easily proved using the formal definition of expectation (see Exercise 1.5).

1.3.2 Statistical Difference vs Bias of all XORs

The following lemma is presented for two reasons, which are similar to those cited in Section 1.3.1:[2] Firstly, this result is also very useful. Secondly, its proof reveals the great benefit that can be obtained by looking at things in a non-obvious way; specifically, by looking at probability distributions (over $\{0,1\}^n$) as functions (from $\{0,1\}^n$ to the reals), and looking at the latter as (2^n-dimensional) vectors.

The following lemma relates two measures of the "randomness" of distributions over n-bit long strings.

- The statistical difference from uniform; namely, the statistical difference (variation difference) between the "target" distribution and the uniform distribution over the set of all n-bit strings.
- The maximum bias of the xor of certain bit positions; namely, the bias of a 0-1 random variable obtained by taking the exclusive-or of certain bits in the "target" distribution.

We show that the statistical difference from uniform is upper-bounded by $\sqrt{2^n}$ times the maximum bias of the xor's, and that this upper bound is tight in general.

The proof of this assertion is based on viewing probability distributions over $\{0,1\}^n$ as elements of a 2^n-dimensional vector space (i.e., the vector space of all functions from $\{0,1\}^n$ to the reals). Furthermore, it turns out that the statistical difference from uniform and the maximum bias of the xor's correspond to two norms defined over this vector space, which are related via the dimention (i.e., 2^n). This connection will be detailed after formally dening all notions.

1.3.2.1 Formal Setting

Let π be a an arbitrary probability distribution over $\{0,1\}^n$ and let μ denote the uniform distribution over $\{0,1\}^n$ (i.e., $\mu(x) = 2^{-n}$ for every $x \in \{0,1\}^n$). Let $x = x_1 \cdots x_n$ and $N \stackrel{\text{def}}{=} 2^n$. The XOR-Lemma relates two "measures of closeness" of π and μ.

[2] Actually, this lemma too has an interesting computational analogue (see [42, Lem. 2.5.8]), alas a less famous one.

- The statistical difference ("variation difference") between π and μ; namely,

$$\mathrm{stat}(\pi) \stackrel{\mathrm{def}}{=} \frac{1}{2} \cdot \sum_x |\pi(x) - \mu(x)| \tag{1.11}$$

- The "maximum bias" of the exclusive-or of certain bit positions in strings chosen according to the distribution π; namely,

$$\mathrm{maxbias}(\pi) \stackrel{\mathrm{def}}{=} \max_{I \neq \emptyset} \{|\pi(\{x : \bigoplus_{i \in I} x_i = 0\}) - \pi(\{x : \bigoplus_{i \in I} x_i = 1\})|\} \tag{1.12}$$

  where for every set $S \subseteq \{0,1\}^n$, we let $\pi(S)$ denote $\sum_{x \in S} \pi(x)$. Indeed each term in Eq. (1.12) can be written as $\mathsf{E}_{(x_1,\ldots,x_n)\sim\pi}[(-1)^{\sum_{i \in I} x_i}]$, where $x \sim \pi$ denotes that x is distributed according to π.

  Note that $\mathrm{maxbias}(\mu) = 0$, since for every $I \neq \emptyset$ it holds that $|\{x : \bigoplus_{i \in I} x_i = 0\}| = 2^{n-1}$.

A simplified version of the XOR-Lemma states that $\mathrm{stat}(\pi) \leq N \cdot \mathrm{maxbias}(\pi)$. The proof is based on viewing distributions as elements in an N-dimensional vector space and observing that the two measures considered by the lemma are merely two norms taken with respect to two different orthogonal bases (see Section 1.3.2.2). Hence, the XOR-Lemma follows from a (more general and quite straightforward) technical lemma that relates norms taken with respect to different orthonormal bases (see Section 1.3.2.3). It turns out this argument actually yields $\mathrm{stat}(\pi) \leq \sqrt{N} \cdot \mathrm{maxbias}(\pi)$, and the inferior bound (i.e., $\mathrm{stat}(\pi) \leq N \cdot \mathrm{maxbias}(\pi)$) is due to a less careful use of the same underlying ideas.

1.3.2.2 Preliminaries: the XOR-Lemma and vector spaces

Probability distributions over $\{0,1\}^n$ are functions from $\{0,1\}^n$ to the reals. Such functions form a N-dimensional vector space, where $N = 2^n$. We shall consider two alternative bases of this vector space.

Recall that vectors (functions) in such a vector space are represented in various bases by considering their inner product with the basis vectors. Thus, the function $f : \{0,1\}^n \to \mathbb{R}$ is represented in the basis $B = \{b_i : i \in [N]\}$ as

$$f(x) = \sum_{i=1}^{2^n} \langle f, b_i \rangle \cdot b_i(x) \tag{1.13}$$

where $\langle f, b_i \rangle = \sum_{z \in \{0,1\}^n} f(z) b_i(z)$ denotes of the inner product of f and b_i.

The standard basis, denoted K, is the orthonormal basis defined by the Kroniker functions; that is, the Boolean functions $\{k_\alpha : \alpha \in \{0,1\}^n\}$, where $k_\alpha(x) = 1$ if $x = \alpha$ and $k_\alpha(x) = 0$ otherwise. Indeed, $\langle f, k_\alpha \rangle = f(\alpha)$, and the statistical difference between two distributions equals (half) the norm L_1 of their difference taken in the above K basis (i.e., $\sum_{x \in \{0,1\}^n} |f(x) - g(x)| = \sum_{x \in \{0,1\}^n} |\langle f - g, k_x \rangle|$).

On the other hand, as we shall see, the maxbias of a distribution equals the maximum Fourier coefficient of the distribution, which in turn corresponds to the max-norm (norm

L_∞) of the distribution taken in a different basis. This basis is defined by the functions $\{b_I : I \subseteq \{1,2,...,n\}\}$, where $b_I(x) = (-1)^{\sum_{i\in I} x_i}$. Note that $b_I(x) = 1$ if the exclusive-or of the bits $\{x_i : i \in I\}$ is 0 and $b_I(x) = -1$ otherwise (e.g., $b_\emptyset \equiv 1$). The new basis is orthogonal but not orthonormal (i.e., $\langle b_I, b_J\rangle = \sum_{x\in\{0,1\}^n}(-1)^{\sum_{i\in I\oplus J} x_i} = 0$ if $I \neq J$ and $\langle b_I, b_I\rangle = N$).[3] Hence, we consider the normalized basis, denoted F, consisting of the functions $f_I = \frac{1}{\sqrt{N}} \cdot b_I$ (where $N = 2^n$). Indeed, the function $g : \{0,1\}^n \to \mathbb{R}$ is represented in this basis via the coefficients $\langle g, f_I\rangle$ (for all $I \subseteq [n]$), and $\langle g, f_I\rangle = \sum_x g(x)b_I(x)/\sqrt{N}$ equals $\sum_x g(x)(-1)^{\sum_{i\in I} x_i}/\sqrt{N}$.

Notation: Let B be an orthonormal basis and r an integer. We denote by $\mathtt{N}_r^B(v)$ the norm L_r of v with respect to the basis B. Namely, $\mathtt{N}_r^B(v) = (\sum_{e\in B} |\langle e, v\rangle|^r)^{(1/r)}$, where $\langle e, v\rangle$ denotes of the inner product of the vectors e and v. We denote by $\mathtt{N}_\infty^B(v)$ the limit of $\mathtt{N}_r^B(v)$ when $r \to \infty$ (i.e., $\mathtt{N}_\infty^B(v)$ is $\max_{e\in B}\{|\langle e, v\rangle|\}$).

We now relate two of the foregoing norms to the measures of closeness presented in Section 1.3.2.1. Thus, Fact 1.2 reduces the XOR-Lemma to relating two norms, which refer to two different orthonormal bases.

Fact 1.2 $\mathrm{stat}(\pi) = \frac{1}{2} \cdot \mathtt{N}_1^K(\pi - \mu)$ *whereas* $\mathrm{maxbias}(\pi) = \sqrt{N} \cdot \mathtt{N}_\infty^F(\pi - \mu)$.

The first equality is immediate by the definitions. Following is a proof of the second equality. Let $\delta(x) = \pi(x) - \mu(x)$. Since $\mathrm{maxbias}(\mu) = 0$, we have $\mathrm{maxbias}(\pi) = \mathrm{maxbias}(\delta)$. Also $\sum_x \delta(x) = 0$, which implies that $\sum_x f_\emptyset(x)\delta(x)$. We get

$$\begin{aligned}
\mathrm{maxbias}(\delta) &= \max_{I\neq\emptyset}\{|\delta(\{x : b_I(x)=1\}) - \delta(\{x : b_I(x)=-1\})|\} \\
&= \max_{I\neq\emptyset}\left\{\left|\sum_x b_I(x)\cdot\delta(x)\right|\right\} \\
&= \sqrt{N}\cdot\max_I\left\{\left|\sum_x f_I(x)\cdot\delta(x)\right|\right\} \\
&= \sqrt{N}\cdot\max_I\{|\langle f_I, \delta\rangle|\} \\
&= \sqrt{N}\cdot\mathtt{N}_\infty^F(\delta)
\end{aligned}$$

We now turn to the actual proof of the XOR Lemma.

1.3.2.3 Proof of the XOR-Lemma

The XOR-Lemma follows from the following technical lemma.

[3] Here we let $I \oplus J$ denote the symmetric difference of the sets I and J. Note that for any non-empty set K it holds that $\sum_{x\in\{0,1\}^n}(-1)^{\sum_{i\in K} x_i}$ equals $\sum_{x\in\{0,1\}^n}\prod_{i\in K}(-1)^{x_i} = 2^{n-|K|}\cdot\prod_{i\in K}\sum_{x_i\in\{0,1\}}(-1)^{x_i}$.

Technical Lemma: *For every two orthogonal bases A and B and every vector v, it holds that*

$$\mathtt{N}_1^A(v) \leq N \cdot \mathtt{N}_\infty^B(v). \tag{1.14}$$

This technical lemma has a three line proof:

For every orthogonal basis A,

$$\mathtt{N}_1^A(v) \leq \sqrt{N} \cdot \mathtt{N}_2^A(v). \tag{1.15}$$

For every pair of orthonormal bases A and B,

$$\mathtt{N}_2^A(v) = \mathtt{N}_2^B(v). \tag{1.16}$$

For every orthogonal basis B,

$$\mathtt{N}_2^B(v) \leq \sqrt{N} \cdot \mathtt{N}_\infty^B(v) \tag{1.17}$$

Indeed, the Technical Lemma (i.e., Eq. (1.14)) is obtained by combining Eq. (1.15)–(1.17). Next, using this Technical Lemma, we get:

Lemma 1.3 (The XOR-Lemma, revised): $\mathrm{stat}(\pi) \leq \frac{1}{2} \cdot \sqrt{N} \cdot \mathrm{maxbias}(\pi)$.

Proof: By Eq. (1.14) (and Fact 1.2), we have

$$\mathrm{stat}(\pi) = \frac{1}{2} \cdot \mathtt{N}_1^K(\pi - \mu) \leq \frac{1}{2} \cdot N \cdot \mathtt{N}_\infty^F(\pi - \mu) = \frac{1}{2} \cdot \sqrt{N} \cdot \mathrm{maxbias}(\pi).$$

■

1.3.2.4 Discussion

The inferior bound, $\mathrm{stat}(\pi) \leq N \cdot \mathrm{maxbias}(\pi)$, has been derived by using one of the following two bounds instead of our Technical Lemma:

1. $\mathtt{N}_1^A(v) \leq \sqrt{N}\mathtt{N}_1^B(v) \leq \sqrt{N} \cdot N\mathtt{N}_\infty^B(v)$.

   The first inequality is proved similarly to the proof of our Technical Lemma (i.e., using $\mathtt{N}_2^B(v) \leq \mathtt{N}_1^B(v)$ instead of Eq. (1.17)). The second inequality is trivial. Each of the two inequalities is tight, but their combination is wasteful.

2. $\mathtt{N}_1^A(v) \leq N \cdot \mathtt{N}_\infty^A(v) \leq N \cdot \sqrt{N}\mathtt{N}_\infty^B(v)$.

   The second inequality is proved similarly to the proof of our Technical Lemma (i.e., using $\mathtt{N}_\infty^A(v) \leq \mathtt{N}_2^A(v)$ instead of Eq. (1.15)). The first inequality is trivial. Again, each of the inequalities is tight, but their combination is wasteful.

Variants. Using small variations on the foregoing argument, we obtain the following variants of the XOR-Lemma:

1. $\max_{x \in \{0,1\}^n}\{|\pi(x) - \mu(x)|\} \leq \mathrm{maxbias}(\pi)$.
2. $\mathrm{stat}(\pi) \leq \sqrt{\sum_{I \neq \emptyset} \mathrm{bias}_I(\pi)^2}$, where $\mathrm{bias}_I(\pi) = \sum_x b_I(x) \cdot \pi(x)$.

Proof: The first claim follows by using $\mathtt{N}_\infty^A(v) \leq \mathtt{N}_2^A(v)$ (instead of $\mathtt{N}_1^A(v) \leq \sqrt{N} \cdot \mathtt{N}_2^A(v)$), and obtaining $\mathtt{N}_\infty^K(\pi - \mu) \leq \sqrt{N} \cdot \mathtt{N}_\infty^F(\pi - \mu)$. The second claim follows by using $\mathtt{N}_1^A(v) \leq \sqrt{N} \cdot \mathtt{N}_2^B(v)$ and $\mathtt{N}_2^F(\pi - \mu) = \sqrt{\sum_{I \neq \emptyset} \mathrm{bias}_I(\pi)^2}$. In both parts we also use $\mathrm{bias}_\emptyset(\pi - \mu) = 0$. ■

1.3.2.5 Advanced topic: Generalization to GF(p), for any prime p

The entire treatment can be generalized to distributions over $\mathrm{GF}(p)^n$, for any prime p. In this case, we redefine $N \stackrel{\mathrm{def}}{=} p^n$, and let $\mathrm{stat}(\pi)$ denote the statistical difference between π and the uniform distribution over $\mathrm{GF}(p)^n$ (cf. Eq. (1.11)). Letting ω denote the p^{th} root of unity, we generalize Eq. (1.12) to

$$\begin{aligned}\mathrm{maxbias}(\pi) &\stackrel{\mathrm{def}}{=} \max_{\beta\in\mathrm{GF}(p)^n\setminus\{0\}^n}\left\{\left|\sum_{e\in\mathrm{GF}(p)}\omega^e\cdot\pi\left(\left\{x:\textstyle\sum_{i\in[n]}\beta_i x_i\equiv e \pmod p\right\}\right)\right|\right\}\\ &= \max_{\beta\in\mathrm{GF}(p)^n\setminus\{0\}^n}\left\{\left|\mathsf{E}_{(x_1,\ldots,x_n)\sim\pi}\left[\omega^{\sum_{i\in[n]}\beta_i x_i}\right]\right|\right\},\end{aligned}$$

where $x\sim\pi$ denotes that x is distributed according to π. The Fourier basis is generalized analogously: The new basic consists of the functions $\{b_\beta:\beta\in\mathrm{GF}(p)^n\}$, where $b_\beta(x)=\omega^{\sum_{i\in[n]}\beta_i x_i}$. The normalized basis, denoted F, consists of the functions $f_\beta=N^{-1/2}\cdot b_\beta$.

Note that, in the case of $p=2$, these definitions coincides with the definitions presented before. By following exactly the same manipulations as in the case of $p=2$, we obtain the following generalization.

The XOR-Lemma, generalized to GF(p): *Let π be an arbitrary distribution over $\mathrm{GF}(p)^n$, and let μ denote the uniform distribution over $\mathrm{GF}(p)^n$. Then*

1. $\mathrm{stat}(\pi)\leq\frac{1}{2}\cdot\sqrt{N}\cdot\mathrm{maxbias}(\pi)$.
2. $\max_{x\in\{0,1\}^n}\{|\pi(x)-\mu(x)|\}\leq\mathrm{maxbias}(\pi)$.
3. $\mathrm{stat}(\pi)\leq\frac{1}{2}\cdot\sqrt{\sum_{\beta\neq0^n}\mathrm{bias}_\beta(\pi)^2}$, *where* $\mathrm{bias}_\beta(\pi)=\sum_x b_\beta(x)\cdot\pi(x)$.

Notes

The three probabilistic inequalities discussed in Section 1.2 appear in almost any standard textbook on probability theory as well as in any textbook that is aimed at randomized computation.

The first XOR Lemma (which refers to the XOR of all variables) is a special case of a famous information theoretic bservation, which asserts that some functions of a message (e.g., the XOR of its bits) that is sent over a noisy channel are pracitically impossible to recover. This lemma is the information theoretic analogue of Yao's XOR Lemma, which plays a central role in complexity theory and is much harder to prove (see [47] or [44, §7.2.1.2]).

The second XOR lemma (which relates the statistical difference to the maximum bias of some XOR of bits (see Section 1.3.2)) is commonly attributed to Umesh Vazirani [111].[4] The proof presented here has appeared as an appendix in [8].

[4] The special case in which the maxbias is zero appears in Chor *et. al.*—[32].

Exercises

Exercise 1.1 (the two forms of statistical difference) Prove the equality stated in Eq. (1.1); that is,

$$\frac{1}{2} \cdot \sum_v |\mathsf{Pr}[X = v] - \mathsf{Pr}[Y = v]| \;=\; \max_S \{\mathsf{Pr}[X \in S] - \mathsf{Pr}[Y \in S]\}.$$

Guideline: Consider the set $S = \{v : \mathsf{Pr}[X = v] > \mathsf{Pr}[Y = v]\}$.

Exercise 1.2 (basic properties of expectation) Prove the following facts:

1. For any two random varaibles X and Y (not necessarilty independent ones) and any two reals $a, b \in \mathbb{R}$, it holds that $\mathsf{E}(aX + bY) = a \cdot \mathsf{E}(X) + b \cdot \mathsf{E}(Y)$.
2. For every random varaible X, it holds that $\mathsf{Pr}[X \geq \mathsf{E}(X)] > 0$.
3. For every random varaible X, it holds that $\mathsf{E}((X - \mathsf{E}(X))^2) = \mathsf{E}(X^2) - \mathsf{E}(X)^2$.
4. For every random varaible X and real $a \in \mathbb{R}$, it holds that $\mathsf{Var}(aX) = a^2 \cdot \mathsf{Var}(X)$.

Guideline: In all cases, just start from the formal definition and apply some elementary algebraic maipulations.

Exercise 1.3 (using linearity of expectation) Prove that for every 3CNF formula, there exists a truth assignment that satisfies at least $7/8$ of the clauses.

Guideline: Consider a uniformly distributed truth assignment, and define $X_i = 1$ if this assignment satisfies the i^{th} clause. Next, lower-bound $\mathsf{E}[\sum_i X_i]$, and use $\mathsf{Pr}[Y \geq \mathsf{E}(Y)] > 0$.

Exercise 1.4 (Markov Inequality, variants) Prove the following inequality both by a reduction to Markov Inequality and by a direct argument. Let X be a random variable such that $\mathsf{Pr}[X > b] = 0$ and $\mu = \mathsf{E}(X)$. Then, for every $v < \mu$, it holds that $\mathsf{Pr}\left[X \leq v\right] \leq \frac{b-\mu}{b-v}$. Show that, for every $\varepsilon > 0$, it holds that $\mathsf{Pr}\left[X > \mu - \varepsilon\right] \geq \frac{\varepsilon}{b-\mu+\varepsilon}$.

Exercise 1.5 (basic properties of independent random variables) Prove the following two facts:

1. Let $X_1, ..., X_n$ be totally independent random variables and f be an arbitrary function defined on their support. Then, the random variables $f(X_1), ..., f(X_n)$ are totally independent.
2. For any two independent random varaibles X and Y, it holds that $\mathsf{E}[XY] = \mathsf{E}[X] \cdot \mathsf{E}[Y]$.

Guideline: In all cases, just start from the formal definition and apply some elementary algebraic maipulations.

Exercise 1.6 (k-wise independent sampling) Prove that for any k-wise independent random variables $X_1, X_2, ..., X_n$ such that $\mu = \sum_{i=1}^{n} \mathsf{E}[X_i]/n$ and $\max_{i\in[n]}\{\mathsf{Var}[X_i]\} \leq 1$, and any $\varepsilon > 0$, it holds that

$$\mathsf{Pr}\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - \mu\right| \geq \varepsilon\right] < \left(\frac{2k}{\varepsilon^2 n}\right)^{\lfloor k/2 \rfloor}$$

Guideline: It is, again, a good idea to consider the random variables $\overline{X}_i \stackrel{\text{def}}{=} X_i - \mathsf{E}(X_i)$, which are k-wise independent. Generalize Chebyshev's inequality and the succesive corollary (regarding pairwise independent sampling) by considering the k^{th} power of $\sum_{i=1}^{n} X_i/n$.

$$\begin{aligned}\mathsf{Pr}\left[\left|\sum_{i=1}^{n}\frac{X_i}{n} - \mu\right| \geq \varepsilon\right] &\leq \frac{\mathsf{E}\left[\left(\sum_{i=1}^{n}\overline{X}_i\right)^k\right]}{\varepsilon^k \cdot n^k} \\ &= \frac{1}{\varepsilon^k \cdot n^k} \cdot \sum_{i_1,\ldots,i_k\in[n]} \mathsf{E}\left[\prod_{j=1}^{k}\overline{X}_{i_j}\right].\end{aligned}$$

The crucial observation is that of all the n^k terms, *the only non-zero terms are those in which no variable occurs exactly once.* This is the case since (by k-wise independence) each term $\mathsf{E}\left[\prod_{j=1}^{k}\overline{X}_{i_j}\right]$ equals $\prod_{i\in I}\mathsf{E}\left[\overline{X}_i^{e_i}\right]$, where $I = \{i_j : j \in [k]\}$ and $e_i \stackrel{\text{def}}{=} |\{j : i_j = i\}|$. (Indeed, $\mathsf{E}\left[\overline{X}_i^{e_i}\right] = 0$ if $e_i = 1$, and $\mathsf{E}\left[\overline{X}_i^{e_i}\right] \leq \mathsf{E}\left[\overline{X}_i^2\right]$ if $e_1 \geq 2$.) Upper-bounding the number of terms in which each occurring variable appears at least twice, we obtain the desired bound. Specifically, show that the number of such terms is at most $\binom{n}{k/2} \cdot (k/2)^k < n^{k/2} \cdot (2k)^{k/2}$.

Lecture 2

The Probabilistic Method

The current text is highly tentative. It includes several applications of the Probabilistic Method, where each of these applications is in an area that deserves a much more comprehensive treatement. In one case (i.e., error correcting codes), we found it hard to refrain from providing a brief overview of the area, whereas in another case (i.e., expander graphs) we refer the reader to the treatment provided in Lecture 5. We mention that a comprehensive treatment of the probabilistic method is provided in Alon and Spencer's book *The Probabilistic Method* [9].

Loosely speaking, the probabilistic method consists of proving the existence of objects that have certain properties by *defining a probability distribution on objects and showing that objects drawn according to this distribution have the desired property with positive probability.* Often, the definition of the probability distribution is simple, but sometimes it is quite complex. Likewise, often objects drawn according to this distribution have the desired property with overwhelmingly high probability, but in other cases they have the property only with small non-zero probability. Several examples follow.[1]

2.1 Graphs of High Degree with no Large Cliques

While it is obvious that there exist graphs that have no large cliques, this fact is less obvious when one insists on a minimum vertex degree. The existence of such graphs can be proved easily using the probabilistic method.

Proposition 2.1 (on the existence of graphs of high degree with no large cliques): *For all sufficiently large N, there exists an N-vertex graph of minimum vertex degree $N/2$ and maximal clique size at most $2\log_2 N + 2$.*

Proof: Consider a random process in which an N-vertex graph is constructed by connecting each pair of vertices with probability $p = 0.5+N^{-1/3}$, independently of all other pairs. Then, with overwhelmingly high probability, each vertex in the resulting has degree greater than

[1] We mention that an even simpler example appears in Exercise 1.3.

$N/2$ (and we may consider an N-vertex subgraph of minimum vertex degree $N/2$). On the other hand, the probability that this random graph has a clique of size k is at most

$$p^{\binom{k}{2}} \cdot \binom{N}{k} < p^{k(k-1)/2} \cdot N^k/(k!). \tag{2.1}$$

Letting $k = 2\log_2 N + 3$, we obtain an expression that tends to zero, which means that a random graph is unlikely to have a k-clique. We conclude that there exists an N-vertex graph that satisfies the degree requirement and has no clique of size $2\log_2 N + 3$. ■

Digest. Indeed, in the proof of Proposition 2.1, we refrained from using the uniform distribution over all graphs only because we aimed at graphs in which the minimum vertex degree equals $N/2$ (see also Exercise 2.1). Still, the distribution used is simple enough. Furthermore, we did show that the desired property holds with overwhelmingly high probability (over the distribution considered).

2.2 Polynomial-Size Monotone Formula for Majority

This section provides a proof of the existence of polynomial-size monotone formula for Majority. The exposition follows the main principles of Valiant's proof (cf. [108]), but deviates from it in some details.

While it is easy to construct quasi-polynomial-size monotone formulae for majority (by relying on divide-and-conquer approaches)[2], it is less obvious how to construct polynomial-size formulae (let alone monotone ones; cf. [108] and the references there-in).

Notation. Suppose, for simplicity that n is odd, and consider the majority function $\mathtt{MAJ} : \{0,1\}^n \to \{0,1\}$ defined as $\mathtt{MAJ}(x) = 1$ if $\mathtt{wt}(x) > n/2$ and $\mathtt{MAJ}(x) = 0$ otherwise, where $\mathtt{wt}(x) = \{i \in [n] : x_i = 1\}$ denotes the Hamming weight of $x = x_1 \cdots x_n$.

Theorem 2.2 *There exist polynomial-size monotone formulae for computing majority.*

The existence of polynomial-size (monotone) formulae is known to be equivalent to the existence of logarithmic-depth (monotone) circuits of bounded fan-in.[3] Anyhow, we shall prove the existence of logarithmic-depth monotone formulae (of bounded fan-in) for majority. Actually, two radically different proofs are known: The first proof uses a rather complicated construction of sorting networks of logarithmic depth [4, 86].[4] The second proof, presented below, uses the probabilistic method.

[2] One way is using the recursion $\mathtt{TH}_t(x'x'') = \vee_{i=0}^{t}(\mathtt{TH}_i(x') \wedge \mathtt{TH}_{t-i}(x''))$, where $\mathtt{TH}_t(z) = 1$ iff $\mathtt{wt}(z) \geq t$. Using $\mathtt{MAJ}(x) = \mathtt{TH}_{|x|/2}(x)$, this yields a size recursion of the form $S(n) = O(n) \cdot S(n/2)$, which solves to $S(n) = O(n)^{\log_2 n}$.

[3] One direction is almost trivial, for the other direction see [98].

[4] Sorting networks may be viewed as Boolean circuits with bit-comparison gates (a.k.a comperators), where each comperator is a (2-bit) sorting device. Observe that a comparator can be implemented by a small monotone circuit (i.e., $\mathtt{comp}(x,y) = (\min(x,y), \max(x,y)) = ((x \wedge y), (x \vee y))$), and that the middle bit of the sorted sequence equals the majority value.

Proof: We prove the existence of logarithmic-depth monotone formulae (of bounded fan-in) for majority. The proof proceeds in two main steps. The first step consists of reducing the worst-case problem (i.e., of computing $\mathtt{MAJ}$ on all inputs) to an average-case problem, denoted Π, where the point of the reduction is that it seems easier to cope with random inputs (than with all possible inputs). Specifically, we shall use a (simple) randomized reduction of the computation of $\mathtt{MAJ}(x)$ to the computation of $\Pi(R(x))$, where $R(x)$ denotes the output of the reduction on input x. The key observation is that if the error probability is sufficiently low (i.e., lower than $2^{-|x|}$), then this randomized reduction yields a non-uniform reduction that is correct on all inputs. (Hence the existence of such a non-uniform reduction is proved by using the probabilistic method.) Next (i.e., in the second step), we show that a very simple (monotone) formula suffices for solving Π on random instances. Finally, composing the (monotone) reduction with the latter formula, we obtain the desired (monotone) formula.

We start with the randomized reduction. Specifically, given an n-bit long input $x = x_1 \cdots x_n$, we consider a sequence of independent identically distributed 0-1 random variables $R(x) = (y_1, ..., y_m)$ such that for every $j \in [m]$ an index $i \in [n]$ is selected uniformly and y_j is set to x_i. Thus, $\mathsf{Pr}[y_i = 1] = \mathtt{wt}(x)/n$. Now, let $F : \{0,1\}^m \to \{0,1\}$ be an arbitrary function. The key observation is captured by the following fact.

Fact 2.3 *If, for every $x \in \{0,1\}^n$, it holds that $\mathsf{Pr}[F(R(x)) = \mathtt{MAJ}(x)] > 1 - 2^{-n}$, then there exists a choice of coin tosses ω for the random process R such that for every $x \in \{0,1\}^n$ it holds that $F(R_\omega(x)) = \mathtt{MAJ}(x)$, where R_ω denotes the deterministic function obtained by fixing the coins of R to ω. Furthermore, for every fixed ω, the function R_ω just projects its input bits to fixed locations in its output sequence.*

(Here the probabilistic method is used to infer the existence of ω such that $F \circ R_\omega = \mathtt{MAJ}$, based on $\mathsf{Pr}_\omega[(\forall x \in \{0,1\}^n)\ F(R_\omega(x)) = \mathtt{MAJ}(x)] > 0$.) Note that, by the furthermore-clause, $F \circ R_\omega$ preserves the complexity and monotonicity of F.

Regarding the feasibility of the hypothesis of Fact 2.3 (i.e., $\mathsf{Pr}[F(R(x)) = \mathtt{MAJ}(x)] > 1 - 2^{-n}$ for every x), consider the case that $m = \Theta(n^3)$ and $F = \mathtt{MAJ}$. (This is merely a sanity check; we cannot afford using this F, because this would reduce the problem to itself on longer input length.) In this case, for every x it holds that

$$\mathsf{Pr}[\mathtt{MAJ}(R(x)) = \mathtt{MAJ}(x)] > \mathsf{Pr}\left[\left|\frac{\mathtt{wt}(R(x))}{m} - \frac{\mathtt{wt}(x)}{n}\right| < \frac{1}{2n}\right], \tag{2.2}$$

which indeed is at least $1 - 2^{-n}$ (by Chernoff bound).

We now turn to the second step, which consists of presenting a monotone formula F of $O(\log n)$ depth. Generalizing the foregoing hypothesis, we wish F to satisfy the following condition: *If $Y_1, ..., Y_m$ are independent identically distributed 0-1 random variables such that for some b it holds that $\mathsf{Pr}[Y_1 = b] \geq 0.5 + 1/2n$, then $\mathsf{Pr}[F(Y_1, ..., Y_m) = b] > 1 - 2^{-n}$.* (For simplicity, we assume that m is a power of three.)

The construction uses a full ternary tree of depth $\ell = \log_3 m$, where internal vertices compute the majority of their three children. Specifically, let $\mathtt{MAJ}_3$ denote the three-variable majority function. Then, $F_1(z_1, z_2, z_3) = \mathtt{MAJ}_3(z_1, z_2, z_3)$ and for every $i \geq 1$

$$F_{i+1}(z_1, ..., z_{3^{i+1}}) = \mathtt{MAJ}_3(F_i(z_1, ..., z_{3^i}), F_i(z_{3^i+1}, ..., z_{3^i+3^i}), F_i(z_{2 \cdot 3^i+1}, ..., z_{3^{i+1}})). \tag{2.3}$$

Finally, we let $F(z_1,...,z_m) = F_\ell(z_1,...,z_m)$.

The intuition is that each level in F amplifies the bias of the corresponding random variables (i.e., functions of $Y_1,...,Y_m$) towards the majority value. This amplification is due to the amplifiaction property of $\mathtt{MAJ}_3$, which is stated next.

Fact 2.4 *Let Z_1, Z_2, Z_3 be three independent identically distributed 0-1 random variables, and let $p \stackrel{\text{def}}{=} \mathsf{Pr}[Z_1=1]$. Then:*

1. $p' \stackrel{\text{def}}{=} \mathsf{Pr}[\mathtt{MAJ}_3(Z_1,Z_2,Z_3)=1] = 3(1-p)p^2 + p^3$.
2. *Letting $\delta \stackrel{\text{def}}{=} p - 0.5$, it holds that $p' = 0.5 + (1.5 - 2\delta^2)\cdot\delta$.*
3. $p' < 3p^2$.

The three parts of the foregoing fact follow by straightforward calculations.[5]

The second part of Fact 2.4 asserts that if $p = 0.5+\delta > 0.5$ and $\delta \leq \delta_0 < 0.5$, then $p' \geq 0.5 + (1.5 - 2\delta_0^2)\cdot\delta$, which means that the bias (i.e., $p-0.5$) increases by a multiplicative factor in each iteration (until it exceeds δ_0). (Note that we assumed $p \geq 0.5 + 1/2n$, but similar considerations hold for $p \leq 0.5 - 1/2n$.)[6] This means that we can increase the bias from its initial level of at least $1/2n$ to any constant level of $\delta_0 < 1/2$, by using $\ell_1 = c_1 \cdot \log_2(2\delta_0 n)$ iterations of $\mathtt{MAJ}_3$, where $c_1 = 1/\log_2(1.5 - 2\delta_0^2)$.

The best result is obtained by using an arbitrary small $\delta_0 > 0$. In this case, we may use $c_1 \approx 1/\log_2(1.5) \approx 1.70951129$. Using $\ell_2 = O(1)$ additional iterations, we may increase the bias from δ_0 to, say, 0.4.

At this point, we use the third part of Fact 2.4, while considering the probability for a wrong majority value. In each such iteration, this probability is reduced from a current value of $1-p$ to less than $3(1-p)^2$. Thus, using $\ell_3 = \log_2 n$ additional iterations, the probability of a wrong value reduces from $1-(0.5+0.4) < 1/6$ to $3^{2^{\ell_3}-1}\cdot(1/6)^{2^{\ell_3}} < 2^{-2^{\ell_3}} = 2^{-n}$.

Letting $\ell = \ell_1 + \ell_2 + \ell_3 < 2.71\log_2 n$ and $m = 3^\ell$, we obtain a formula $F = F_\ell$ on m variables, but this formula uses the non-standard $\mathtt{MAJ}_3$-gates. Yet, a $\mathtt{MAJ}_3$-gate can be implemented by a depth-three monotone formula (e.g., $\mathtt{MAJ}_3(z_1,z_2,z_3)$ equals $(z_1 \wedge z_2) \vee (z_2 \wedge z_3) \vee (z_3 \wedge z_1)$), and hence F is a monotone formula of depth $3\ell < 8.13\log_2 n$. Note that if $Y_1,...,Y_m$ are independent identically distributed 0-1 random variables such that for some b it holds that $\mathsf{Pr}[Y_1 = b] \geq 0.5 + 1/2n$, then $\mathsf{Pr}[F(Y_1,...,Y_m) = 1-b] < 2^{-n}$. Thus, for every $x \in \{0,1\}^n$ it holds that $\mathsf{Pr}_\omega[F(R_\omega(x)) = 1 - \mathtt{MAJ}(x)] < 2^{-n}$ and $\mathsf{Pr}_\omega[(\forall x \in \{0,1\}^n)\ F(R_\omega(x)) = \mathtt{MAJ}(x)] > 0$ follows. Hence, there exists a choice of ω such that $F \circ R_\omega$ computes the majority of n-bit inputs. ∎

Comment. Interestingly, Valiant [108] obtains a better result by using an iterated construction that uses the function $V(z_1,z_2,z_3,z_4) = (z_1 \vee z_2) \wedge (z_3 \vee z_4)$ as the basic building block (rather than $\mathtt{MAJ}_3$). Since V is not a balanced predicate (i.e., $\mathsf{Pr}[V(U_4)=1] = 9/16$), the random process used in [108] maps the string $x \in \{0,1\}^n$ to a sequence of independent identically distributed 0-1 random variables, $(y_1,...,y_m)$, such that for every $j \in [m]$ the bit

[5] For the second part, use $p' = (3-2p)p^2 = (3-1-2\delta)\cdot(0.25+\delta+\delta^2)$, which implies $p' = 0.5 + 1.5\delta - 2\delta^3$.

[6] One way to see this is to define $p = \mathsf{Pr}[Z_1 = 0]$.

y_j is set to zero with some constant probability β (and is set to x_i otherwise, where $i \in [n]$ is uniformly distributed). The value of β is chosen such that if Z_1, Z_2, Z_3, Z_4 are independent identically distributed 0-1 random variables satisfying $\mathsf{Pr}[Z_1 = 1] = p \stackrel{\text{def}}{=} (1 - \beta)/2$, then $\mathsf{Pr}[V(Z_1, Z_2, Z_3, Z_4) = 1] = p$. It turns out that V amplifies deviation from p slightly better than $\mathtt{MAJ}_3$ does (w.r.t $1/2$).[7] More importantly, V can be implemented by a monotone formula of depth two, whereas $\mathtt{MAJ}_3$ requires depth three. Thus, Valiant [108] performs $2.65 \log_2 n$ iterations (rather than $2.71 \log_2 n$ itertations), and obtains a formula of depth $5.3 \log_2 n$.

2.3 Error Correcting Codes

The application of the probabilistic method is given in the proof of Proposition 2.5, but the current section goes well beyond this example. Indeed, we seize the opportunity to highlight some issues and aspects of coding theory that are most relevant to the theory of computation. The interested reader is referred to [101] for a more comprehensive treatment of the computational aspects of coding theory. Structural aspects of coding theory, which are at the traditional focus of that field, are covered in standard textbook such as [80].

2.3.1 Basic Notions

Loosely speaking, an error correcting code is a mapping of strings to longer strings such that any two different strings are mapped to a corresponding pair of strings that are far apart (and not merely different). Specifically, $C : \{0,1\}^k \to \{0,1\}^n$ is a (binary) code of distance d if for every $x \neq y \in \{0,1\}^k$ it holds that $C(x)$ and $C(y)$ differ on at least d bit positions. Indeed, *the relation between k, n and d is of major concern: typically, the aim is having a large distance* (i.e., large d) *without introducing too much redundancy*[8] (i.e., have n as small as possible with respect to k (and d)).

It will be useful to extend the foregoing definition to sequences over an arbitrary (finite) alphabet Σ, and to use some notations. Specifically, for $x \in \Sigma^m$, we denote the i^{th} symbol of x by x_i (i.e., $x = x_1 \cdots x_m$), and consider codes over Σ (i.e., mappings of Σ-sequences to Σ-sequences). The mapping (code) $C : \Sigma^k \to \Sigma^n$ has distance d if for every $x \neq y \in \Sigma^k$ it holds that $|\{i : C(x)_i \neq C(y)_i\}| \geq d$. The members of $\{C(x) : x \in \Sigma^k\}$ are called codewords (and in some texts this set itself is called a code).

In general, we define a metric, called Hamming distance, over the set of n-long sequences over Σ. The Hamming distance between y and z, where $y, z \in \Sigma^n$, is defined as the number of locations on which they disagree (i.e., $|\{i : y_i \neq z_i\}|$). The Hamming weight of such sequences is defined as the number of non-zero elements (assuming that one element of Σ is viewed as zero). Typically, Σ is associated with an additive group, and in this case the distance between y and z equals the Hamming weight of $w = y - z$, where $w_i = y_i - z_i$ (for every i).

[7] This is surprising only if we forget that V takes four inputs rather than three.

[8] Note that a trivial way of obtaining distance d is to duplicate each symbol d times. This ("repetition") code satisfies $n = d \cdot k$, while we shall seek $n \ll d \cdot k$. Indeed, as we shall see, one can obtain simultaneously $n = O(k)$ and $d = \Omega(k)$.

Asymptotics. We will actually consider infinite families of codes; that is, $\{C_k : \Sigma_k^k \to \Sigma_k^{n(k)}\}_{k\in S}$, where $S \subseteq \mathbb{N}$ (and typically $S = \mathbb{N}$). (N.B., we allow Σ_k to depend on k.) We say that such a family has distance $d : \mathbb{N} \to \mathbb{N}$ if for every $k \in S$ it holds that C_k has distance $d(k)$. Needless to say, both $n = n(k)$ (called the block-length) and $d(k)$ depend on k, and *the aim is having a linear dependence* (i.e., $n(k) = O(k)$ and $d(k) = \Omega(n(k))$). In such a case, one talks of the relative rate of the code (i.e., the constant $k/n(k)$) and its relative distance (i.e., the constant $d(k)/n(k)$). In general, we will often refer to *relative distances* between sequences. For example, for $y, z \in \Sigma^n$, we say that y and z are ε-close (resp., ε-far) if $|\{i : y_i \neq z_i\}| \leq \varepsilon \cdot n$ (resp., $|\{i : y_i \neq z_i\}| \geq \varepsilon \cdot n$).

Explicitness. A mild notion of explicitness refers to constructing the list of all codewords in time that is polynomial in its length (which is exponential in k). A more standard notion of explicitness refers to generating a specific codeword (i.e., producing $C(x)$ when given x), which coincides with the encoding task mentioned next. Stronger notions of explicitness refer to other computational problems concerning codes (e.g., various decoding tasks).

Computational problems. The most basic computational tasks associated with codes are encoding and decoding (under noise). The definition of the encoding task is straightforward (i.e., map $x \in \Sigma_k^k$ to $C_k(x)$), and an efficient algorithm is required to compute each symbol in $C_k(x)$ in $\mathrm{poly}(k, \log|\Sigma_k|)$-time.[9] When defining the decoding task we note that "minimum distance decoding" (i.e., given $w \in \Sigma_k^{n(k)}$, find x such that $C_k(x)$ is closest to w (in Hamming distance)) is just one natural possibility. Two related variants, regarding a code of distance d, are:

Unique decoding: Given $w \in \Sigma_k^{n(k)}$ that is at Hamming distance less than $d(k)/2$ from some codeword $C_k(x)$, retrieve the corresponding decoding of $C_k(x)$ (i.e., retrieve x).

Needless to say, this task is well-defined because there cannot be two different codewords that are each at Hamming distance less than $d(k)/2$ from w.

List decoding: Given $w \in \Sigma_k^{n(k)}$ and a parameter d' (which may be greater than $d(k)/2$), output a list of all codewords (or rather their decoding) that are at Hamming distance at most d' from w. (That is, the task is outputting the list of all $x \in \Sigma_k^k$ such that $C_k(x)$ is at distance at most d' from w.)

Typically, one considers the case that $d' < d(k)$. See Section 2.3.4 for a discussion of upper-bounds on the number of codewords that are within a certain distance from a generic sequence.

Two additional computational tasks are considered in Section 2.3.3.

[9] The foregoing formulation is not the one that is common in coding theory, but it is the most natural one for our applications. On one hand, this formulation is applicable also to codes with super-polynomial block-length. On the other hand, this formulation does not support a discussion of practical algorithms that compute the codeword faster than is possible when computing each of the codeword's bits separately.

Linear codes. Associating Σ_k with some finite field, we call a code $C_k : \Sigma_k^k \to \Sigma_k^{n(k)}$ linear if it satisfies $C_k(x+y) = C_k(x) + C_k(y)$, where x and y (resp., $C_k(x)$ and $C_k(y)$) are viewed as k-dimensional (resp., $n(k)$-dimensional) vectors over Σ_k, and the arithmetic is of the corresponding vector space. A useful property of linear codes is that their distance equals the Hamming weight of the lightest codeword other than $C_k(0^k)$ $(= 0^{n(k)})$; that is, $\min_{x \neq y}\{|\{i : C_k(x)_i \neq C_k(y)_i\}|\}$ equals $\min_{x \neq 0^k}\{|\{i : C_k(x)_i \neq 0\}|\}$. Another useful property of linear codes is that the code is fully specified by a k-by-$n(k)$ matrix, called the generating matrix, that consists of the codewords of some fixed basis of Σ_k^k. That is, the set of all codewords is obtained by taking all $|\Sigma_k|^k$ different linear combination of the rows of the generating matrix.

2.3.2 A Few Popular Codes

Our focus will be on explicitly constructible codes; that is, (families of) codes of the form $\{C_k : \Sigma_k^k \to \Sigma_k^{n(k)}\}_{k \in S}$ that are coupled with efficient encoding and decoding algorithms. But before presenting several such codes, let us consider a non-explicit code (having "good parameters"); that is, the following result asserts the existence of certain codes without pointing to any specific code (let alone an explicit one).

Proposition 2.5 (on the distance of random linear codes): *Let $n, d, t : \mathbb{N} \to \mathbb{N}$ be such that, for all sufficiently large k, it holds that*

$$n(k) \;\geq\; \max\left(2d(k),\, \frac{k + t(k)}{1 - H_2(d(k)/n(k))}\right), \tag{2.4}$$

where $H_2(\alpha) \stackrel{\text{def}}{=} \alpha \log_2(1/\alpha) + (1-\alpha)\log_2(1/(1-\alpha))$ is the binary entropy function.[10] *Then, for all sufficiently large k, with probability greater than $1 - 2^{-t(k)}$, a random linear transformation of $\{0,1\}^k$ to $\{0,1\}^{n(k)}$ constitutes a code of distance $d(k)$.*

Indeed, for asserting that most random linear codes are good it suffices to set $t = 1$, while for merely asserting the existence of a good linear code even setting $t = 0$ will do. Also, *for every constant $\delta \in (0, 0.5)$ there exists a constant $\rho > 0$ and an infinite family of codes $\{C_k : \{0,1\}^k \to \{0,1\}^{k/\rho}\}_{k \in \mathbb{N}}$ of relative distance δ.* Specifically, any constant $\rho \geq (1 - H_2(\delta))$ will do.

Proof: We consider a uniformly selected k-by-$n(k)$ generating matrix over GF(2), and upper-bound the probability that it yields a linear code of distance less than $d(k)$. We use a union bound on all possible $2^k - 1$ linear combinations of the rows of the generating matrix, where for each such combination we compute the probability that it yields a codeword of Hamming weight less than $d(k)$. Observe that the result of each such linear combination is uniformly distributed over $\{0,1\}^{n(k)}$, and thus this codeword has Hamming weight less than $d(k)$ with probability $p \stackrel{\text{def}}{=} \sum_{i=0}^{d(k)-1} \binom{n(k)}{i} \cdot 2^{-n(k)}$. Clearly, for $d(k) \leq n(k)/2$, it holds that $p < d(k) \cdot 2^{-(1-H_2(d(k)/n(k)))\cdot n(k))}$, but actually $p \leq 2^{-(1-H_2(d(k)/n(k)))\cdot n(k))}$ holds as well (e.g., use [9, Cor. 14.6.3]). Using $(1 - H_2(d(k)/n(k))) \cdot n(k) \geq k + t(k)$, the proposition follows. ■

[10] Recall that the entropy of a random variable X is defined as $\sum_x \mathsf{Pr}[X = x] \cdot \log_2(1/\mathsf{Pr}[X = x])$. Thus, $H_2(\alpha)$ equals the entropy of a 0-1 random variable that is 1 with probability α.

2.3.2.1 A mildly explicit version of Proposition 2.5

Note that Proposition 2.5 yields a deterministic algorithm that finds a linear code of distance $d(k)$ by conducting an exhaustive search over all possible generating matrices; that is, a good code can be found in time $\exp(k \cdot n(k))$. The time bound can be improved to $\exp(k + n(k))$, by constructing the generating matrix in iterations such that, at each iteration, the current set of rows is augmented with a single row while maintaining the natural invariance (i.e., all non-empty linear combinations of the current rows have weight at least $d(k)$). Thus, at each iteration, we conduct an exhaustive search over all possible values of the next ($n(k)$-bit long) row, and for each such candidate value we check whether the foregoing invariance holds (by considering all linear combinations of the previous rows and the current candidate).

To analyze the foregoing algorithm, note that the proof of Proposition 2.5 can be adapted to assert that, as long as we have less than k rows, a random choice of the next row will do with positive probability. Thus, the foregoing iterative algorithm finds a good code in time $\sum_{i=1}^{k} 2^{n(k)} \cdot 2^{i-1} \cdot \mathrm{poly}(n(k)) = \exp(n(k) + k)$. In the case that $n(k) = O(k)$, this yields an algorithm that runs in time that is polynomial in the size of the code (i.e., the number of codewords (i.e., 2^k)). Needless to say, this mild level of explicitness is inadequate for most coding applications; however, it will be useful to us in Section 2.3.2.5.

2.3.2.2 The Hadamard Code

The Hadamard code is the longest (non-repetitive) *linear* code over $\{0,1\} \equiv \mathrm{GF}(2)$. That is, $x \in \{0,1\}^k$ is mapped to the sequence of all $n(k) = 2^k$ possible linear combinations of its bits; that is, bit locations in the codewords are associated with k-bit strings such that location $\alpha \in \{0,1\}^k$ in the codeword of x holds the value $\sum_{i=1}^{k} \alpha_i x_i$. It can be verified that each non-zero codeword has weight 2^{k-1}, and thus this code has relative distance $d(k)/n(k) = 1/2$ (albeit its block-length $n(k)$ is exponential in k).

Turning to the computational aspects, we note that encoding is very easy. As for decoding, the warm-up discussion at the beginning of the proof of [44, Thm. 7.7] provides a very fast probabilistic algorithm for unique decoding, whereas [44, Thm. 7.8] itself provides a very fast probabilistic algorithm for list decoding.

We mention that the Hadamard code has played a key role in the proof of the PCP Theorem (see [11, 12] or [44, Thm. 9.16]).

A propos long codes. We mention that the longest (non-repetitive) binary code (called the Long-Code and introduced in [19]) is extensively used in the design of "advanced" PCP systems (see, e.g., [56, 57]). In this code, a k-bit long string x is mapped to the sequence of $n(k) = 2^{2^k}$ values, each corresponding to the evaluation of a different Boolean function at x; that is, bit locations in the codewords are associated with Boolean functions such that the location associated with $f : \{0,1\}^k \to \{0,1\}$ in the codeword of x holds the value $f(x)$.

2.3.2.3 The Reed–Solomon Code

Reed-Solomon codes can be defined for any adequate non-binary alphabet, where the alphabet is associated with a finite field of n elements, denoted $\mathrm{GF}(n)$. For any $k < n$, the code maps univariate polynomials of degree $k-1$ over $\mathrm{GF}(n)$ to their evaluation at all field

elements. That is, $p \in \mathrm{GF}(n)^k$ (viewed as such a polynomial), is mapped to the sequence $(p(\alpha_1), ..., p(\alpha_n))$, where $\alpha_1, ..., \alpha_n$ is a canonical enumeration of the elements of $\mathrm{GF}(n)$.[11] This mapping is called a Reed-Solomon code with parameters k and n, and its distance is $n - k + 1$ (because any non-zero polynomials of degree $k - 1$ evaluates to zero at less than k points). Indeed, this code is linear (over $\mathrm{GF}(n)$), since $p(\alpha)$ is a linear combination of $p_0, ..., p_{k-1}$, where $p(\zeta) = \sum_{i=0}^{k-1} p_i \zeta^i$.

The Reed-Solomon code yields infinite families of codes with constant rate and constant relative distance (e.g., by taking $n(k) = 3k$ and $d(k) = 2k$), but the alphabet size grows with k (or rather with $n(k) > k$). Efficient algorithms for unique decoding and list decoding are known (see [100] and references therein). These computational tasks correspond to the extrapolation of polynomials based on a noisy version of their values at all possible evaluation points.

2.3.2.4 The Reed–Muller Code

Reed-Muller codes generalize Reed-Solomon codes by considering multi-variate polynomials rather than univariate polynomials. Consecutively, the alphabet may be any finite field, and in particular the two-element field GF(2). Reed-Muller codes (and variants of them) are extensively used in complexity theory; for example, they underly hardness amplification schemes (see, e.g., [44, Const. 7.11]) and some PCP constructions (e.g., [12, 11]). The relevant property of these (non-binary) codes is that, under a suitable setting of parameters that satisfies $n(k) = \mathrm{poly}(k)$, they allow super fast "codeword testing" and "self-correction" (see discussion in Section 2.3.3).

For any prime power q and parameters m and r, we consider the set, denoted $P_{m,r}$, of all m-variate polynomials of *total degree* at most r over $\mathrm{GF}(q)$. Each polynomial in $P_{m,r}$ is represented by the $k = \log_q |P_{m,r}|$ coefficients of all relevant monomials, where in the case that $r < q$ it holds that $k = \binom{m+r}{m}$. We consider the code $C : \mathrm{GF}(q)^k \to \mathrm{GF}(q)^n$, where $n = q^m$, mapping m-variate polynomials of total degree at most r to their values at all q^m evaluation points. That is, the m-variate polynomial p of total degree at most r is mapped to the sequence of values $(p(\overline{\alpha}_1), ..., p(\overline{\alpha}_n))$, where $\overline{\alpha}_1, ..., \overline{\alpha}_n$ is a canonical enumeration of all the m-tuples of $\mathrm{GF}(q)$. The relative distance of this code is lower-bounded by $(q - r)/q$ (cf. Section 9.1.3).

In typical applications one sets $r = \Theta(m^2 \log m)$ and $q = \mathrm{poly}(r)$, which yields $k > m^m$ and $n = \mathrm{poly}(r)^m = \mathrm{poly}(m^m)$. Thus we have $n(k) = \mathrm{poly}(k)$ but not $n(k) = O(k)$. As we shall see in Section 2.3.3, the advantage (in comparison to the Reed-Solomon code) is that codeword testing and self-correction can be performed at complexity related to $q = \mathrm{poly}(\log n)$. Actually, most complexity applications use a variant in which only m-variate polynomials of *individual degree* $r' = r/m$ are encoded. In this case, an alternative presentation (analogous to the one presented in Footnote 11) is preferred: The information is viewed as a function $f : H^m \to \mathrm{GF}(q)$, where $H \subset \mathrm{GF}(q)$ is of size $r' + 1$, and is encoded by the evaluation at all points in $\mathrm{GF}(q)^m$ of the (unique) m-variate polynomial of individual degree r' that extends the function f (see [44, Const. 7.11]).

[11] Alternatively, we may map $(v_1, ..., v_k) \in \mathrm{GF}(n)^k$ to $(p(\alpha_1), ..., p(\alpha_n))$, where p is the unique univariate polynomial of degree $k - 1$ that satisfies $p(\alpha_i) = v_i$ for $i = 1, ..., k$. Note that this modification amounts to a linear transformation of the generating matrix.

2.3.2.5 Binary codes of constant relative distance and constant rate

Recall that we seek binary codes of constant relative distance and constant rate. Proposition 2.5 asserts that such codes exists, but does not provide an explicit construction. The Hadamard code is explicit but does not have a constant rate (to say the least (since $n(k) = 2^k$)).[12] The Reed-Solomon code has constant relative distance and constant rate but uses a non-binary alphabet (which grows at least linearly with k). Thus, all codes we have reviewed so far fall short of providing an *explicit construction of binary codes of constant relative distance and constant rate*. We achieve the desired construction by using the paradigm of concatenated codes [38], which is of independent interest. (Concatenated codes may be viewed as a simple analogue of the proof composition paradigm of [12] (cf. [44, §9.3.2.2]).)

Intuitively, concatenated codes are obtained by first encoding information, viewed as a sequence over a large alphabet, by some code and next encoding each resulting symbol, which is viewed as a sequence of over a smaller alphabet, by a second code. Formally, consider $\Sigma_1 \equiv \Sigma_2^{k_2}$ and two codes, $C_1 : \Sigma_1^{k_1} \to \Sigma_1^{n_1}$ and $C_2 : \Sigma_2^{k_2} \to \Sigma_2^{n_2}$. Then, the concatenated code of C_1 and C_2, maps $(x_1, ..., x_{k_1}) \in \Sigma_1^{k_1} \equiv \Sigma_2^{k_1 k_2}$ to $(C_2(y_1), ..., C_2(y_{n_1}))$, where $(y_1, ..., y_{n_1}) = C_1(x_1, ..., x_{k_1})$.

Note that the resulting code $C : \Sigma_2^{k_1 k_2} \to \Sigma_2^{n_1 n_2}$ has constant rate and constant relative distance if both C_1 and C_2 have these properties. Encoding in the concatenated code is straightforward. To decode a corrupted codeword of C, we view the input as an n_1-long sequence of blocks, where each block is an n_2-long sequence over Σ_2. Applying the decoder of C_2 to each block, we obtain n_1 sequences (each of length k_2) over Σ_2, and interpret each such sequence as a symbol of Σ_1. Finally, we apply the decoder of C_1 to the resulting n_1-long sequence (over Σ_1), and interpret the resulting k_1-long sequence (over Σ_1) as a $k_1 k_2$-long sequence over Σ_2. The key observation is that *if $w \in \Sigma_2^{n_1 n_2}$ is $\varepsilon_1\varepsilon_2$-close to $C(x_1, ..., x_{k_1}) = (C_2(y_1), ..., C_2(y_{n_1}))$ then at least $(1-\varepsilon_1) \cdot n_1$ of the blocks of w are ε_2-close to the corresponding $C_2(y_i)$.*[13]

We are going to consider the concatenated code obtained by using the Reed-Solomon Code $C_1 : \mathrm{GF}(n_1)^{k_1} \to \mathrm{GF}(n_1)^{n_1}$ as the large code, setting $k_2 = \log_2 n_1$, and using the mildly explicit version of Proposition 2.5 (see also Section 2.3.2.1) $C_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ as the small code. We use $n_1 = 3k_1$ and $n_2 = O(k_2)$, and so the concatenated code is $C : \{0,1\}^k \to \{0,1\}^n$, where $k = k_1 k_2$ and $n = n_1 n_2 = O(k)$. The key observation is that C_2 can be constructed in $\exp(k_2)$-time, whereas here $\exp(k_2) = \mathrm{poly}(k)$. Furthermore, both encoding and decoding with respect to C_2 can be performed in time $\exp(k_2) = \mathrm{poly}(k)$. Thus, we get:

Theorem 2.6 (an explicit good code): *There exists constants $\delta, \rho > 0$ and an explicit family of binary codes of rate ρ and relative distance at least δ. That is, there exists a polynomial-time* (encoding) *algorithm C such that $|C(x)| = |x|/\rho$* (for every x) *and a polynomial-time* (decoding) *algorithm D such that for every y that is $\delta/2$-close to some $C(x)$ it holds that $D(y) = x$. Furthermore, C is a linear code.*

[12] Binary Reed-Muller codes also fail to simultaneously provide constant relative distance and constant rate.

[13] This observation offers unique decoding from a fraction of errors that is the product of the fractions (of error) associated with the two original codes. Stronger statements regarding unique decoding of the concatenated code can be made based on more refined analysis (cf. [38]).

The linearity of C is justified by using a Reed-Solomon code over the extension field $F = \mathrm{GF}(2^{k_2})$, and noting that this code induces a linear transformation over GF(2). Specifically, the value of a polynomial p over F at a point $\alpha \in F$ can be obtained as a linear transformation of the coefficient of p, when viewed as k_2-dimensional vectors over GF(2).

Relative distance approaching one half. Note that starting with a Reed-Solomon code of relative distance δ_1 and a smaller code C_2 of relative distance δ_2, we obtain a concatenated code of relative distance $\delta_1\delta_2$. Recall that, for any constant $\delta_1 < 1$, there exists a Reed-Solomon code $C_1 : \mathrm{GF}(n_1)^{k_1} \to \mathrm{GF}(n_1)^{n_1}$ of relative distance δ_1 and constant rate (i.e., $1 - \delta_1$). Thus, for any constant $\varepsilon > 0$, we may obtain an explicit code of constant rate and relative distance $(1/2) - \varepsilon$ (e.g., by using $\delta_1 = 1 - (\varepsilon/2)$ and $\delta_2 = (1 - \varepsilon)/2$). Furthermore, giving up on constant rate, we may start with a Reed-Solomon code of block-length $n_1(k_1) = \mathrm{poly}(k_1)$ and distance $n_1(k_1) - k_1$ over $[n_1(k_1)]$, and use a Hadamard code (encoding $[n_1(k_1)] \equiv \{0,1\}^{\log_2 n_1(k_1)}$ by $\{0,1\}^{n_1(k_1)}$) in the role of the small code C_2. This yields a (concatenated) binary code of block length $n(k) = n_1(k)^2 = \mathrm{poly}(k)$ and distance $(n_1(k) - k) \cdot n_1(k)/2$. Thus, *the resulting explicit code has relative distance* $\frac{1}{2} - \frac{k}{2\sqrt{n(k)}} = \frac{1}{2} - o(1)$*, provided that* $n(k) = \omega(k^2)$.

2.3.3 Advanced Topic: Two Additional Computational Problems

In this section we briefly review relaxations of two traditional coding theoretic tasks. The purpose of these relaxations is enabling the design of super-fast (randomized) algorithms that provide meaningful information. Specifically, these algorithms may run in sub-linear (e.g., poly-logarithmic) time, and thus cannot possibly solve the unrelaxed version of the corresponding problem.

Local testability. This task refers to testing whether a given word is a codeword (in a predetermine code), based on (randomly) inspecting few locations in the word. Needless to say, we can only hope to make an approximately correct decision; that is, accept each codeword and reject with high probability each word that is *far* from the code. (Indeed, this task is within the framework of property testing; see [92].)

Local decodability. Here the task is to recover a specified bit in the plaintext by (randomly) inspecting few locations in a mildly corrupted codeword. This task is somewhat related to the task of self-correction (i.e., recovering a specified bit in the codeword itself, by inspecting few locations in the mildly corrupted codeword).

Note that the Hadamard code is both locally testable and locally decodable as well as self-correctable (based on a constant number of queries into the word). However, the Hadamard code has an exponential block-length (i.e., $n(k) = 2^k$), and the question is whether one can achieve analogous results with respect to a shorter code (e.g., $n(k) = \mathrm{poly}(k)$). As hinted in Section 2.3.2.4, the answer is positive (when we refer to performing these operations in time that is poly-logarithmic in k):

Theorem 2.7 *For some constant $\delta > 0$ and polynomials $n, q : \mathbb{N} \to \mathbb{N}$, there exists an explicit family of codes $\{C_k : [q(k)]^k \to [q(k)]^{n(k)}\}_{k\in\mathbb{N}}$ of relative distance δ that can be locally*

testable and locally decodable in $\mathrm{poly}(\log k)$*-time. That is, the following three conditions hold.*

1. Encoding: *There exists a polynomial time algorithm that on input* $x \in [q(k)]^k$ *returns* $C_k(x)$.

2. Local Testing: *There exists a probabilistic polynomial-time oracle machine* T *that given* k (in binary)[14] *and oracle access to* $w \in [q(k)]^{n(k)}$ (viewed as $w:[n(k)]\to[q(k)]$) *distinguishes the case that* w *is a codeword from the case that* w *is* $\delta/2$*-far from any codeword. Specifically:*

   (a) *For every* $x \in [q(k)]^k$ *it holds that* $\mathsf{Pr}[T^{C_k(x)}(k)=1] = 1$.

   (b) *For every* $w \in [q(k)]^{n(k)}$ *that is* $\delta/2$*-far from any codeword of* C_k *it holds that* $\mathsf{Pr}[T^w(k)=1] \leq 1/2$.

   As usual, the error probability can be reduced by repetitions.

3. Local Decoding: *There exists a probabilistic polynomial-time oracle machine* D *that given* k *and* $i \in [k]$ (in binary) *and oracle access to any* $w \in [q(k)]^{n(k)}$ *that is* $\delta/2$*-close to* $C_k(x)$ *returns* x_i*; that is,* $\mathsf{Pr}[D^w(k,i)=x_i] \geq 2/3$.

   Self correction *holds too: there exists a probabilistic polynomial-time oracle machine* M *that given* k *and* $i \in [n(k)]$ (in binary) *and oracle access to any* $w \in [q(k)]^{n(k)}$ *that is* $\delta/2$*-close to* $C_k(x)$ *returns* $C_k(x)_i$*; that is,* $\mathsf{Pr}[D^w(k,i)=C_k(x)_i] \geq 2/3$.

We stress that all these oracle machines work in time that is polynomial in the binary representation of k, which means that they run in time that is poly-logarithmic in k. The code asserted in Theorem 2.7 is a (small modification of a) Reed-Muller code, for $r = m^2 \log m < q(k) = \mathrm{poly}(r)$ and $[n(k)] \equiv \mathrm{GF}(q(k))^m$ (see Section 2.3.2.4).[15] The aforementioned oracle machines queries the oracle $w:[n(k)]\to\mathrm{GF}(q(k))$ at a non-constant number of locations. Specifically, self-correction for location $i \in \mathrm{GF}(q(k))^m$ is performed by selecting a random line (over $\mathrm{GF}(q(k))^m$) that passes through i, recovering the values assigned by w to all $q(k)$ points on this line, and performing univariate polynomial extrapolation (under mild noise). Local testability is easily reduced to self-correction, and (under the aforementioned modification) local decodability is a special case of self-correction.

Constant number of (binary) queries. The local testing and decoding algorithms asserted in Theorem 2.7 make a polylogarithmic number of queries into the oracle. Furthermore, these queries (which refer to a non-binary code) are non-binary (i.e., they are each answered by a non-binary value). In contrast, the Hadamard code has local testing and decoding algorithms that use a *constant number of binary queries*. Can this be obtained with much shorter (binary) codewords? That is, redefining local testability and decodability as requiring a *constant number of queries*, we ask whether binary codes of significantly shorter block-length can be locally testable and decodable. For local testability the answer

[14] Thus, the running time of T is $\mathrm{poly}(|k|) = \mathrm{poly}(\log k)$.

[15] The modification is analogous to the one presented in Footnote 11: For a suitable choice of k points $\overline{\alpha}_1, ..., \overline{\alpha}_k \in \mathrm{GF}(q(k))^m$, we map $v_1, ..., v_k$ to $(p(\overline{\alpha}_1), ..., p(\overline{\alpha}_n))$, where p is the unique m-variate polynomial of degree at most r that satisfies $p(\overline{\alpha}_i) = v_i$ for $i = 1, ..., k$.

is definitely positive: one can construct such (locally testable and binary) codes with block-length that is nearly linear (i.e., linear up to polylogarithmic factors; see [25, 35]). For local decodability, the shortest known code has super-polynomial length (see [114]). In light of this state of affairs, we advocate natural relaxations of the local decodability task (e.g., the one studied in [24]).

The interested reader is referred to [43], which includes more details on locally testable and decodable codes as well as a wider perspective. (Note, however, that this survey was written prior to [35] and [114], which resolve two major open problems discussed in [43].)

2.3.4 Advanced Topic: A List Decoding Bound

A necessary condition for the feasibility of the list decoding task is that the list of codewords that are close to the given word is short. In this section we present an upper-bound on the length of such lists, noting that this bound has found several applications in complexity theory (and specifically to studies related to the contents of this book). In contrast, we do not present far more famous bounds (which typically refer to the relation among the main parameters of codes (i.e., k, n and d)), because they seem less relevant to the contents of this book.

We start with a general statement that refers to any alphabet $\Sigma \equiv [q]$, and later specialize it to the case that $q = 2$. Especially in the general case, it is natural and convenient to consider the agreement (rather than the distance) between sequences over $[q]$. Furthermore, it is natural to focus on agreement rate of at least $1/q$, and it is convenient to state the following result in terms of the "excessive agreement rate" (i.e., the excess beyond $1/q$).[16] Loosely speaking, the following result upper-bounds the number of codewords that have a (sufficient) large agreement rate with any fixed sequence, where the upper-bound depends only on this agreement rate and the agreement rate between codewords (as well as on the alphabet size, but not on k and n).

Lemma 2.8 (Part 2 of [48, Thm. 15]): *Let $C : [q]^k \to [q]^n$ be an arbitrary code of distance $d \leq n - (n/q)$, and let $\eta_{\mathrm{C}} \stackrel{\mathrm{def}}{=} (1 - (d/n)) - (1/q) \geq 0$ denote the corresponding upper-bound on the excessive agreement rate between codewords. Suppose that $\eta \in (0, 1)$ satisfies*

$$\eta > \sqrt{\left(1 - \frac{1}{q}\right) \cdot \eta_{\mathrm{C}}} \tag{2.5}$$

Then, for any $w \in [q]^n$, the number of codewords that agree with w on at least $((1/q) + \eta) \cdot n$ positions (i.e., are at distance at most $(1 - ((1/q) + \eta)) \cdot n$ from w) *is upper-bounded by*

$$\frac{(1 - (1/q))^2 - (1 - (1/q)) \cdot \eta_{\mathrm{C}}}{\eta^2 - (1 - (1/q)) \cdot \eta_{\mathrm{C}}} \tag{2.6}$$

In the binary case (i.e., $q = 2$), Eq. (2.5) requires $\eta > \sqrt{\eta_{\mathrm{C}}/2}$ and Eq. (2.6) yields the upper-bound $(1 - 2\eta_{\mathrm{C}})/(4\eta^2 - 2\eta_{\mathrm{C}})$. We highlight two specific cases:

[16] Indeed, we only consider codes with distance $d \leq (1 - 1/q) \cdot n$ (i.e., agreement rate of at least $1/q$) and words that are at distance at most d from the code. Note that a random sequence is expected to agree with any fixed sequence on a $1/q$ fraction of the locations.

1. At the end of Section 8.2.2 we refer to this bound (for the binary case) while setting $\eta_{\mathrm{C}} = (1/k)^2$ and $\eta = 1/k$. Indeed, in this case $(1-2\eta_{\mathrm{C}})/(4\eta^2 - 2\eta_{\mathrm{C}}) = O(k^2)$.

2. In the case of the Hadamard code, we have $\eta_{\mathrm{C}} = 0$. Thus, for every $w \in \{0,1\}^n$ and every $\eta > 0$, the number of codewords that are $(0.5-\eta)$-close to w is at most $1/4\eta^2$.

In the general case (and specifically for $q \gg 2$) it is useful to simplify Eq. (2.5) by $\eta > \min\{\sqrt{\eta_{\mathrm{C}}}, (1/q) + \sqrt{\eta_{\mathrm{C}} - (1/q)}\}$ and Eq. (2.6) by $\frac{1}{\eta^2 - \eta_{\mathrm{C}}}$.

2.4 Expander Graphs

In this section we prove the existence of 3-regular expanders, where expander graphs are defined and discussed in Lecture 5. Specifically, we say that a graph $G = (V, E)$ is ε-expanding if for every vertex set S of cardinality at most $|V|/2$ it holds that $|\Gamma_G(S)| \geq (1+\varepsilon) \cdot |S|$, where $\Gamma_G(S) \stackrel{\mathrm{def}}{=} \{v : \exists u \in S \text{ s.t. } \{u,v\} \in E\}$ denote the set of vertices that neighbor some vertex in S.

Theorem 2.9 (on the existence of 3-regular expanders): *For a sufficiently small $\varepsilon > 0$ and all sufficiently large N there exists a 3-regular N-vertex graph that is ε-expanding.*

Proof: It is actually easier to prove the related statement that refers to the alternative definition of combinatorial expansion that refers to the relative size of $\Gamma^+_G(S) = \Gamma_G(S) \setminus S$ (rather than to the relative size of $\Gamma_G(S)$). That is, we shall first prove that, for a sufficiently small $\varepsilon > 0$ and all sufficiently large N, a random 3-regular N-vertex graph is "ε-expanding" (i.e., $|\Gamma^+_G(S)| \leq \varepsilon|S|$ for every S of size at most $N/2$) with overwhelmingly high probability. The proof proceeds by considering a (not necessarily simple) graph G obtained by combining three uniformly chosen perfect matchings of the elements of $[N]$. For every $S \subseteq [N]$ of size at most $N/2$ we consider the probability that $|\Gamma_G(S) \setminus S| \leq \varepsilon|S|$. This probability is upper-bounded by

$$\sum_{T_1,T_2,T_3 \subset [N]:\, (\forall i)\, |T_i| \leq \varepsilon|S|} p_{S,T_1} \cdot p_{S,T_2} \cdot p_{S,T_3}, \tag{2.7}$$

where $p_{S,T}$ denotes the probability that for a random perfect matching M it holds that every element in $S' \stackrel{\mathrm{def}}{=} S \cup T$ is matched by M to an element in S'. (The point is that $|\Gamma_G(S) \setminus S| \leq \varepsilon|S|$ implies that there exists T_1, T_2, T_3 of size at most $\varepsilon|S|$ such that the i^{th} matching pairs $S \cup T_i$ with itself.) Viewing each random matching as induced by pairing consecutive elements in a random ordering of $[N]$ and letting $K = |S'|$, we consider the $K/2$ pairs that contain elements of S', and get

$$\begin{aligned} p_{S,T} &= \frac{\binom{N/2}{K/2} \cdot K! \cdot (N-K)!}{N!} \\ &= \frac{\binom{N/2}{K/2}}{\binom{N}{K}} \\ &< \binom{N}{K}^{-1/2}. \end{aligned}$$

The argument is concluded by applying a union bound. Specifically, the number of tuples (S, T_1, T_2, T_3) that we need to consider equals $\binom{N}{s} \cdot \binom{N}{\varepsilon s}^3 \approx \binom{N}{s}^{1+3\varepsilon}$, where s denotes the size of S. Thus, we get an upper bound of $\sum_{s \leq N/2} \binom{N}{s}^{1+3\varepsilon} \cdot \binom{N}{(1+\varepsilon)s}^{-3/2}$, which tends to zero with N.

We now turn to the actual claim of the theorem, which is proved by considering a 3-regular graph obtained by combining an N-cycle with a random matching of the first $N/2$ vertices and the remaining $N/2$ vertices. As a warm-up, we first establish the existence of d-regular expanders, for some constant d. In particular, foreseeing the case of $d = 3$, consider a random graph G on the vertex set $V = \{0, ..., 2n-1\}$ constructed by augmenting the fixed edge set $\{\{i, i+1 \bmod n\} : i=0, ..., n-1\} \cup \{\{n+i, n+(i+1 \bmod n)\} : i=0, ..., n-1\}$ with $d-2$ uniformly (and independently) chosen perfect matchings of the vertices of $L \stackrel{\text{def}}{=} \{0, ..., n-1\}$ to the vertices of $R \stackrel{\text{def}}{=} \{n, ..., 2n-1\}$. (See Figure 2.1.) That is, we start with two fixed n-cycles (one traversing all vertices in L and the other traversing all vertices in R) with $d-2$ random matchings of L to R. For a sufficiently small universal constant $\varepsilon > 0$, we upper-bound the probability that such a random graph is not ε-expanding.

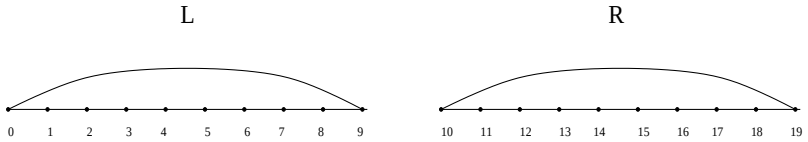

Figure 2.1: The fixed cycle edges in the random expander (with $n = 10$).

Let us denote by $\Gamma_C(S)$ the vertices that are adjacent to vertices in S via (the fixed) cycle edges; that is, $\Gamma_C(S \cap L) = \{i \pm 1 \bmod n : i \in S \cap L\}$ and $\Gamma_C(S \cap R) = \{n + (i \pm 1 \bmod n) : i \in S \cap R\}$. Similarly, we denote by $\Gamma_M(S)$ the vertices that are adjacent to vertices in S via the random matching edges; that is, $\Gamma_M(S \cap L) = \Gamma_G(S \cap L) \setminus \Gamma_C(S \cap L)$, which equals $\Gamma_G(S \cap L) \cap R$, and similarly for $S \cap R$. We note that, for every set S, it holds that $|\Gamma_G(S \cap L) \cap L| \geq |S \cap L|$ (and similarly for R), because $\Gamma_C(S \cap L) \subseteq \{i+1 \bmod n : i \in S \cap L\}$. Let us also assume that $|\Gamma_C(S) \leq (1+\varepsilon) \cdot |S|$, since we are done with the set S otherwise. Thus, we focus on the sizes of $\Gamma_M(S \cap L) \setminus \Gamma_C(S \cap R)$ and $\Gamma_M(S \cap R) \setminus \Gamma_C(S \cap L)$, which are the contributions to $\Gamma_G(S)$ that are due to the random matchings of L to R. Fixing a set S of size at most $n = |V|/2$, we upper-bound the probability that the foregoing contribution is smaller than $\varepsilon|S|$. Actually, assuming without loss of generality that $|S \cap L| \geq |S \cap R|$, we upper-bound the probability that $|\Gamma_M(S \cap L) \setminus \Gamma_C(S \cap R)| < \varepsilon|S|$. The latter probability is upper-bounded by p_S^{d-2}, where p_S denotes the probability that a uniformly selected matching of L to R matches $S \cap L$ to a set that contains less than $\varepsilon|S|$ elements in $R \setminus \Gamma_C(S \cap R)$. That is,

$$\begin{aligned} p_S &\stackrel{\text{def}}{=} \sum_{i=0}^{\varepsilon|S|-1} \frac{\binom{|R|-|\Gamma_C(S\cap R)|}{i} \cdot \binom{|\Gamma_C(S\cap R)|}{|S\cap L|-i}}{\binom{|R|}{|S\cap L|}} \\ &< \frac{\binom{n-\ell}{\varepsilon|S|} \cdot \binom{\ell+\varepsilon|S|}{|S\cap L|}}{\binom{n}{|S\cap L|}} \end{aligned}$$

where $\ell = |\Gamma_C(S \cap R)|$. Indeed, we may focus on the case that $|S \cap L| \leq \ell + \varepsilon|S|$ (because in the other case $p_S = 0$),[17] and observe that for every $\alpha < 1/2$ (e.g., $\alpha = 1/3$) there exists a sufficiently small $\varepsilon > 0$ such that $p_S < \binom{2n}{|S|}^{-\alpha}$ (since in this case $p_S < \binom{n}{|S|/2}^{-(1-5\varepsilon)}$).[18] The claim follows for $d \geq 5$, by using a union bound on all sets S of size at most n (i.e., $\sum_{S:|S|\leq n} \binom{2n}{|S|} \cdot p_S^{d-2}$ vanishes, since $p_S \ll \binom{2n}{|S|}^{-1/3}$).

To deal with the case $d = 3$, we use a more sophisticated union bound. Intuitively, we partition the elements of S to elements that occur on relatively long arithmetic subsequences (with a step increment of either 1 or 2) and to the rest. The number of sets that have many elements of the first type is relatively small (and so we gain in the union bound), whereas elements of the second type contribute a significant expansion via the cycle edges. Details follow.

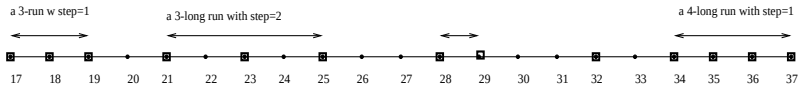

Figure 2.2: Vertices in the set S''' in the random expander (with $t > 4$).

Fixing an adequate constant t (i.e., $t = 1/\sqrt{\varepsilon}$) and an arbitrary set S, we denote by S' the set of all elements of S that reside in t-long intervals of S; that is, S' contains i if there exist $s \in \{1, ..., t\}$ such that $\{i+j-s : j = 1, ..., t\} \subseteq S$. Next, we place in S'' all elements of $S \setminus S'$ that reside in t-long arithmetic sequence with step increment 2; that is, S'' contains i if there exist $s \in \{1, ..., t\}$ such that $\{i+2j-2s : j = 1, ..., t\} \subseteq S \setminus S'$. (See Figure 2.2.) Letting $S''' \stackrel{\text{def}}{=} S \setminus (S' \cup S'')$, we first claim that $|\Gamma_C(S)| > |S| + |S'''|/2t$. This claim is proved by observing that any arithmetic sequence (i.e., of any step increament) has a C-neighborhood greater than itself, whereas S''' has no (step 1 or 2) arithmetic sequence of length t (and the additional units contributed by the various arithmetic sequences are counted at most twice).[19] Thus, if $|S'''| > 2|S|/t$, then $|\Gamma_C(S)| > |S' \cup S''| + (1 + (1/2t)) \cdot |S'''| > (1 + t^{-2}) \cdot |S|$ and $|\Gamma_G(S)| > (1 + \varepsilon) \cdot |S|$ follows (since $t = 1/\sqrt{\varepsilon}$). Hence, it suffices to consider the case

[17] Indeed, if $|S \cap L| > \ell + \varepsilon|S|$, then more than $\varepsilon|S|$ elements of $S \cap L$ are matched to elements in $R \setminus \Gamma_C(S \cap R)$.

[18] We may assume that $\ell = |\Gamma_C(S \cap R)| \leq |S \cap R| + \varepsilon|S|$, since we are done with the set S otherwise (becuase $|\Gamma_G(S)| \geq |\Gamma_C(S \cap R)| + |S \cap L|$). Combining this hypothesis with the foregoing upper bound on p_S, we get

$$\begin{aligned} p_S &< \frac{\binom{n}{\varepsilon|S|} \cdot \binom{|S\cap R|+2\varepsilon|S|}{|S\cap L|}}{\binom{n}{|S\cap L|}} \\ &\leq \frac{\binom{n}{\varepsilon|S|} \cdot \binom{|S\cap L|+2\varepsilon|S|}{2\varepsilon|S|}}{\binom{n}{|S\cap L|}} \end{aligned}$$

where the second inequality is due to $|S \cap L| \geq |S \cap R|$. Recalling that $|S \cap L| \leq \ell + \varepsilon|S| \leq |S \cap R| + 2\varepsilon|S|$, we obtain $p_S < \binom{n}{\varepsilon|S|}^3 / \binom{n}{(1+2\varepsilon)|S|/2}$.

[19] Thus, the argument consists of three observations. The first observation is that, for any $\ell < n$, any arithmetic sequence $A = \{i + s \cdot j : j = 1, .., \ell\}$ satisfies $|\Gamma_C(A)| \geq |A| + 1$. Furthermore, if $s = 1$ and $\ell > 2$, then $|\Gamma_C(A)| = |A| + 2$. Next, we partition S into disjoint maximal arithmetic sequences, denoted $A_1, ..., A_k$, such that arithmetic sequences of step 1 have maximum length (singletons are considered as sequences of step 2). Note that if A_i and A_j cannot neighbor one another (i.e., if A_i ends at p then A_j

$|S'''| \leq 2|S|/t$ and apply the same analysis as before (while uasing a finer counting). Recall that for any fixed S, the probability that $|\Gamma_M(S) \setminus \Gamma_C(S)| \leq \varepsilon \cdot |S|$ is smaller than $\binom{2n}{|S|}^{-1/3}$. On the other hand, the number of possible choices of $S = S' \cup S'' \cup S'''$ of total size s such that $|S'''| \leq 2s/t$ is at most

$$\sum_{i=0}^{2s/t} \binom{2n}{i} \sum_{j=1}^{s/t} \binom{2n-i}{2j} \cdot 2^j \;<\; 2 \cdot \binom{2n}{2s/t}\binom{2n}{2s/t} \cdot 2^{s/t},$$

where $i = |S'''|$ and j denotes the number of arithmetic sequences in $S' \cup S''$ (where each such sequence is determined by the choice of its endpoints and the type of step). Thus, the union bound yields $\sum_{s \leq n} \binom{2n}{2s/t}^2 \cdot 2^{(s/t)+1} \cdot \binom{2n}{s}^{-1/3}$, and we are done (provided t is sufficiently large). ■

Notes

Indeed, the standard application of the argument presented in Proposition 2.1 is for proving that, for sufficiently large N, there exists an N-vertex graph that contains neither a $(2\log_2 N + 3)$-vertex clique nor a $(2\log_2 N + 3)$-vertex independent set (cf., e.g., [9, Sec. 1.1]). We mention that the latter result is relatively tight, since every N-vertex graph has either a $(0.5\log_2 N + 1)$-vertex clique or a $(0.5\log_2 N + 1)$-vertex independent set.[20]

Other applications of the probabilistic method occur in Exercises 3.2 and 3.10 (which assert the existence of certain pseudorandom generators) and in Exercise 8.2 (which assert the existence of certain randomness extractors).

We repeat our warning regarding the highly tentative nature of the current text and refer the reader again to the textbook of Alon and Spencer (*The Probabilistic Method* [9].

Exercises

Exercise 2.1 (generalization of Proposition 2.1) Prove the following statement for $k : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ that is large as you can: For every N and $d < N$, there exists a N-vertex graph of minimum vertex degree d and maximal clique size at most $k(N, d)$.

---

cannot start at $p+1$). Furthermore, if A_i ends at p and A_j starts at $p+2$, then it cannot be that both use step increament 2. Hence, $|\Gamma_C(\bigcup_{i\in[k]} A_i)|$ is lower bounded by $\sum_{i\in[k]} |A_i| + (k/2)$, because in each case where a double accounting occurs it is the case that at least one of the two contributed sequences has step 1. We conclude that $|\Gamma_C(S)| \geq |S| + (k/2)$, where $k \geq |S'''|/t$.

[20] The latter claim follows as a special case of Ramsey's Theorem (cf., e.g., [54]): Let $R(k,\ell)$ be the smallest integer N such that any N-vertex graph has either a k-vertex clique or an ℓ-vertex independent set. Then, $R(k,\ell) \leq R(k-1,\ell) + R(k,\ell-1)$, and $R(k,\ell) \leq 2^{k+\ell-2}$ follows (by also using $R(k,1) = R(1,\ell) = 1$). The inductive claim is proved by picking an arbitrary vertex v in the generic graph $G = (V,E)$, and observing that either $|\Gamma_G(v)| \geq R(k-1,\ell)$ or $|\overline{\Gamma}_G(v)| \geq R(k,\ell-1)$, where $\overline{\Gamma}_G(v) \stackrel{\text{def}}{=} V \setminus (\Gamma_G(v) \cup \{v\})$. If $|\Gamma_G(v)| \geq R(k-1,\ell)$, then either the subgraph induced by $\Gamma_G(v)$ contains an ℓ-vertex indepedent set (and so does G), or the subgraph induced by $\Gamma_G(v)$ contains a $(k-1)$-vertex clique (and so G contains a k-vertex clique). If $|\overline{\Gamma}_G(v)| \geq R(k-1,\ell)$, then either the subgraph induced by $\overline{\Gamma}_G(v)$ contains a k-vertex clique (and so does G), or the subgraph induced by $\overline{\Gamma}_G(v)$ contains an $(\ell-1)$-vertex indepedent set (and so G contains an ℓ-vertex indepedent set).

Lecture 3

Special Purpose Generators

Although our interest in this course is confined to special types of pseudorandom generators, we consider it a good idea to discuss these special cases within the general framework of pseudorandom generators.

3.1 The Wider Picture: A General Paradigm

We advocate a unified view of various notions of pseudorandom generators. That is, we view these notions as incarnations of a general abstract paradigm, to be presented in this section. A reader who is interested only in one of these incarnations may still use this section as a general motivation towards the specific definitions used later. On the other hand, some readers may prefer reading this section after studying one of the specific incarnations.

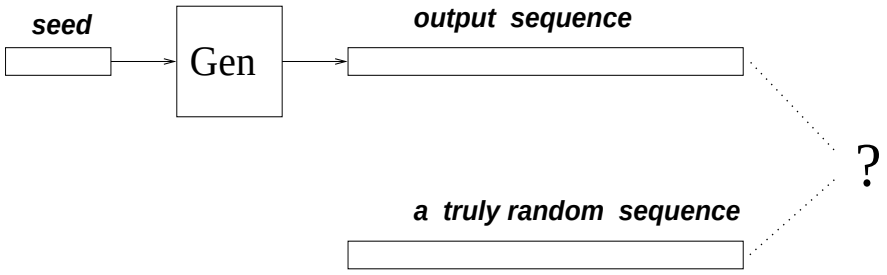

Figure 3.1: Pseudorandom generators – an illustration.

3.1.1 Three fundamental aspects

A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretch measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational*

indistinguishability requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*). Let us elaborate.

Stretch function: A necessary requirement from any notion of a pseudorandom generator is that the generator is a *deterministic algorithm* that stretches short strings, called seeds, into longer output sequences.[1] Specifically, this algorithm stretches k-bit long seeds into $\ell(k)$-bit long outputs, where $\ell(k) > k$. The function $\ell:\mathbb{N}\to\mathbb{N}$ is called the stretch measure (or stretch function) of the generator. In some settings the specific stretch measure is immaterial (e.g., see [44, Sec. 8.2]).

Computational Indistinguishability: A necessary requirement from any notion of a pseudorandom generator is that the generator "fools" some non-trivial algorithms. That is, it is required that any algorithm taken from a predetermined class of interest cannot distinguish the output produced by the generator (when the generator is fed with a uniformly chosen seed) from a uniformly chosen sequence. Thus, we consider a class $\mathcal{D}$ of distinguishers (e.g., probabilistic polynomial-time algorithms) and a class $\mathcal{F}$ of (threshold) functions (e.g., reciprocals of positive polynomials), and require that the generator G satisfies the following: For any $D \in \mathcal{D}$, any $f \in \mathcal{F}$, and for all sufficiently large k it holds that

$$|\,\mathsf{Pr}[D(G(U_k)) = 1] \;-\; \mathsf{Pr}[D(U_{\ell(k)}) = 1]\,| \;<\; f(k)\,, \tag{3.1}$$

where U_n denotes the uniform distribution over $\{0,1\}^n$, and the probability is taken over U_k (resp., $U_{\ell(k)}$) as well as over the coin tosses of algorithm D in case it is probabilistic. The reader may think of such a distinguisher, D, as an observer who tries to tell whether the "tested string" is a random output of the generator (i.e., distributed as $G(U_k)$) or is a truly random string (i.e., distributed as $U_{\ell(k)}$). The condition in Eq. (3.1) requires that D cannot make a meaningful decision; that is, ignoring a negligible difference (represented by $f(k)$), D's verdict is the same in both cases.[2] The archetypical choice is that $\mathcal{D}$ is the set of all probabilistic polynomial-time algorithms, and $\mathcal{F}$ is the set of all functions that are the reciprocal of some positive polynomial.

We note that there is a clear tension between the stretching and the computational indistinguishability conditions. Indeed, as shown in Exercise 3.1, the output of any pseudorandom generator is "statistically distinguishable" from the corresponding uniform distribution. However, there is hope that a restricted class of (computationally bounded) distinguishers cannot detect the (statistical) difference; that is, be fooled by some suitable generators. In fact, placing no computational requirements on the generator (or, alternatively, imposing very mild requirements such as upper-bounding the running-time by a double-exponential

[1] Indeed, the seed represents the randomness that is used in the generation of the output sequences; that is, the randomized generation process is decoupled into a deterministic algorithm and a random seed. This decoupling facilitates the study of such processes.

[2] The class of threshold functions $\mathcal{F}$ should be viewed as determining the class of noticeable probabilities (as a function of k). Thus, we require certain functions (i.e., those presented on the l.h.s of Eq. (3.1)) to be smaller than any noticeable function *on all but finitely many integers*. We call the former functions negligible. Note that a function may be neither noticeable nor negligible (e.g., it may be smaller than any noticeable function on infinitely many values and yet larger than some noticeable function on infinitely many other values).

function), yields "generators" that can fool any subexponential-size circuit family (see Exercise 3.2). However, we are interested in the complexity of the generation process, which is the aspect addressed next.

Complexity of Generation: This aspect refers to the complexity of the generator itself, when viewed as an algorithm. That is, here we refer to the resources used by the generator (e.g., its time and/or space complexity). The archetypical choice is that the generator has to work in polynomial-time (i.e., make a number of steps that is polynomial in the length of its input – the seed). Other choices will be discussed as well.

3.1.2 Notational conventions

We will consistently use k for denoting the length of the seed of a pseudorandom generator, and $\ell(k)$ for denoting the length of the corresponding output. In some cases, this makes our presentation a little more cumbersome, where in these cases it is more natural to focus on a different parameter (e.g., the length of the pseudorandom sequence) and let the seed-length be a function of the latter. However, our choice has the advantage of focusing attention on the fundamental parameter of pseudorandom generation process – the length of the random seed. We note that whenever a pseudorandom generator is used to "derandomize" an algorithm, n will denote the length of the input to this algorithm, and k will be selected as a function of n.

3.1.3 Some instantiations of the general paradigm

Two important instantiations of the notion of pseudorandom generators relate to polynomial-time distinguishers.

1. General-purpose pseudorandom generators correspond to the case where the generator itself runs in polynomial-time and needs to withstand *any probabilistic polynomial-time distinguisher*, including distinguishers that run for more time than the generator. Thus, the same generator may be used safely in any efficient application. (This notion is treated in [44, Sec. 8.2].)

2. In contrast, pseudorandom generators intended for derandomization may run for more time than the distinguisher, which is viewed as a fixed circuit having size that is upper-bounded by a fixed polynomial. (This notion is treated in [44, Sec. 8.3].)

In addition, the general paradigm may be instantiated by focusing on the space-complexity of the potential distinguishers (and the generator), rather than on their time-complexity. Furthermore, one may also consider distinguishers that merely reflect probabilistic properties such as pairwise independence, small-bias, and hitting frequency.

3.1.4 Our focus: special-purpose generators

The two instantiations of pseudorandom generators considered in Section 3.1.3 were aimed at decreasing the amount of randomness utilized by any algorithm of certain time complexity (or even fully derandomizing the corresponding complexity class). Specifically, one often

considers the goal of providing relatively efficient deterministic algorithms for any problem in $\mathcal{BPP}$. In the current text our goal is less ambitious. We only seek to derandomize (or decrease the randomness of) specific algorithms or rather classes of algorithms that use their random bits in certain (restricted) ways. For example, the algorithm's correctness may only require that its sequence of coin tosses (or "blocks" in such a sequence) are pairwise independent. Indeed, the restrictions that we shall consider here have a concrete and "structural" form, rather than the abstract complexity theoretic forms considered in previous chapters.

The aforementioned restrictions induce corresponding classes of very restricted distinguishers, which in particular are much weaker than the classes of distinguishers considered in previous chapters. These very restricted types of distinguishers induce correspondingly weak types of pseudorandom generators (which produce sequences that fool these distinguishers). Still, such generators have many applications (both in complexity theory and in the design of algorithms).

We start with the simplest of these generators: the pairwise independence generator, and its generalization to t-wise independence for any $t \geq 2$. Such generators *perfectly* fool any distinguisher that only observe t locations in the output sequence. This leads naturally to almost pairwise (or t-wise) independence generators, which also fool such distinguishers (albeit non-perfectly). The latter generators are implied by a stronger class of generators, which is of independent interest: the small-bias generators. Small-bias generators fool any linear test (i.e., any distinguisher that merely considers the XOR of some fixed locations in the input sequence). We finally turn to the Expander Random Walk Generator: This generator produces a sequence of strings that hit any dense subset of strings with probability that is close to the hitting probability of a truly random sequence.[3]

Comment regarding our parameterization: To maintain consistency with other notions of pseudorandom generators, we continue to present the generators in terms of the seed length, denoted k. Since this is not the common presentation for most results presented in the sequel, we provide (in footnotes) the common presentation in which the seed length is determined as a function of other parameters.

3.2 Pairwise Independence Generators

Pairwise (resp., t-wise) independence generators fool tests that inspect only two (resp., t) elements in the output sequence of the generator. Such local tests are indeed very restricted, yet they arise naturally in many settings. For example, such a test corresponds to a probabilistic analysis (of a procedure) that only relies on the pairwise independence of certain choices made by the procedure. We also mention that, in some natural range of parameters, pairwise independent sampling is as good as sampling by totally independent sample points (see, e.g., Section 1.2.4).

A *t-wise independence generator of block-length* $b : \mathbb{N} \to \mathbb{N}$ (and stretch function ℓ) is a relatively efficient deterministic algorithm (e.g., one that works in time polynomial in the output length) that expands a k-bit long random seed into a sequence of $\ell(k)/b(k)$ blocks,

[3] Related notions such as samplers, dispersers, and extractors are not treated here. The interested reader is directed to Lecture 6 and Lecture 8, respectively.

each of length $b(k)$, such that any t blocks are uniformly and independently distributed in $\{0,1\}^{t\cdot b(k)}$. That is, denoting the i^{th} block of the generator's output (on seed s) by $G(s)_i$, we require that for every $i_1 < i_2 < \cdots < i_t$ (in $[\ell(k)/b(k)]$) it holds that

$$G(U_k)_{i_1}, G(U_k)_{i_2}, ..., G(U_k)_{i_t} \equiv U_{t\cdot b(k)}. \tag{3.2}$$

We note that this condition holds even if the inspected t blocks are selected adaptively (see Exercise 3.3). In case $t = 2$, we call the generator pairwise independent.

3.2.1 Constructions

In the first construction, we refer to $\mathrm{GF}(2^{b(k)})$, the finite field of $2^{b(k)}$ elements, and associate its elements with $\{0,1\}^{b(k)}$.

Theorem 3.1 (t-wise independence generator):[4] *Let t be a fixed integer and let $b, \ell, \ell' : \mathbb{N}\to\mathbb{N}$ such that $b(k) = k/t$, $\ell'(k) = \ell(k)/b(k) > t$ and $\ell'(k) \le 2^{b(k)}$. Let $\alpha_1, ..., \alpha_{\ell'(k)}$ be fixed distinct elements of the field $\mathrm{GF}(2^{b(k)})$. For $s_0, s_1, ..., s_{t-1} \in \{0,1\}^{b(k)}$, let*

$$G(s_0, s_1, ..., s_{t-1}) \stackrel{\text{def}}{=} \left(\sum_{j=0}^{t-1} s_j\alpha_1^j \ , \ \sum_{j=0}^{t-1} s_j\alpha_2^j \ , ..., \ \sum_{j=0}^{t-1} s_j\alpha_{\ell'(k)}^j \right) \tag{3.3}$$

where the arithmetic is that of $\mathrm{GF}(2^{b(k)})$. Then, G is a t-wise independence generator of block-length b and stretch ℓ.

That is, given a seed that consists of t elements of $\mathrm{GF}(2^{b(k)})$, the generator outputs a sequence of $\ell'(k)$ such elements.

Proof: The proof is based on the observation that, for any fixed $v_0, v_1, ..., v_{t-1}$, the condition $\{G(s_0, s_1, ..., s_{t-1})_{i_j} = v_j\}_{j=0}^{t-1}$ constitutes a system of t linear equations over $\mathrm{GF}(2^{b(k)})$ (in the variables $s_0, s_1, ..., s_{t-1}$) such that the equations are linearly-independent. Thus, linear independence of certain expressions yields statistical independence of the corresponding random variables. Details follow.

For every $i = 1, ..., \ell'(k)$, let $X_i = G(s_0, s_1, ..., s_{t-1})_i \in \mathrm{GF}(2^{b(k)})$ be a random variable representing the distribution of the i^{th} element in the generated sequence, when $s_0, s_1, ..., s_{t-1}$ are uniformly distributed in $\mathrm{GF}(2^{b(k)})$. Our aim is to prove that, for every t fixed sequence of indices $i_1, ..., i_t \in [\ell'(k)]$ and every sequence of t possible values $v_1, ..., v_t \in \mathrm{GF}(2^{b(k)})$, it holds that $\mathsf{Pr}[(\forall j \in [t])\, X_{i_j} = v_j] = 2^{-t\cdot b(k)}$. Using Eq. (3.3), we have

$$\mathsf{Pr}\left[\begin{pmatrix} X_{i_1} \\ X_{i_2} \\ \vdots \\ X_{i_t} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{pmatrix}\right] = \mathsf{Pr}_{s_0,s_1,...,s_{t-1}}\left[\begin{pmatrix} \sum_{j=0}^{t-1} s_j\alpha_{i_1}^j \\ \sum_{j=0}^{t-1} s_j\alpha_{i_2}^j \\ \vdots \\ \sum_{j=0}^{t-1} s_j\alpha_{i_t}^j \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{pmatrix}\right]$$

[4] In the common presentation of this t-wise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and $\ell' \le 2^b$, the seed length is set to $t \cdot b$.

$$
\begin{aligned}
&= \mathsf{Pr}_{s_0,s_1,...,s_{t-1}}\left[\begin{pmatrix} 1 & \alpha_{i_1} & \cdots & \alpha_{i_1}^{t-1} \\ 1 & \alpha_{i_2} & \cdots & \alpha_{i_2}^{t-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{i_t} & \cdots & \alpha_{i_t}^{t-1} \end{pmatrix}\begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{t-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{pmatrix}\right] \\
&= \mathsf{Pr}_{s_0,s_1,...,s_{t-1}}\left[\begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{t-1} \end{pmatrix} = \begin{pmatrix} 1 & \alpha_{i_1} & \cdots & \alpha_{i_1}^{t-1} \\ 1 & \alpha_{i_2} & \cdots & \alpha_{i_2}^{t-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{i_t} & \cdots & \alpha_{i_t}^{t-1} \end{pmatrix}^{-1}\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{pmatrix}\right] \\
&= \frac{1}{|\mathrm{GF}(2^{b(k)})|^t}.
\end{aligned}
$$

Indeed, the crucial observation is that the vectors $(1\,,\,\alpha_{i_1}\,,\,\alpha_{i_1}^2\,,\ldots,\,\alpha_{i_1}^{t-1})$ through $(1\,,\,\alpha_{i_t}\,,\,\alpha_{i_t}^2\,,\ldots,\,\alpha_{i_t}^{t-1})$ are linearly independent (cf. Vandermonde matrix). Hence, we "reduces" the stochastic independence of the random variables X_{i_1} through X_{i_t} to the linear independence of the vectors $(1\,,\,\alpha_{i_1}\,,\,\alpha_{i_1}^2\,,\ldots,\,\alpha_{i_1}^{t-1})$ through $(1\,,\,\alpha_{i_t}\,,\,\alpha_{i_t}^2\,,\ldots,\,\alpha_{i_t}^{t-1})$. ■

A somewhat tedious comment. We warn that Eq. (3.3) does not provide a fully explicit construction (of a generator). What is missing is an explicit representation of $\mathrm{GF}(2^{b(k)})$, which requires an irreducible polynomial of degree $b(k)$ over $\mathrm{GF}(2)$. For specific values of $b(k)$, a good representation does exist; e.g., for $d \stackrel{\mathrm{def}}{=} b(k) = 2 \cdot 3^e$ (with e being an integer), the polynomial $x^d + x^{d/2} + 1$ is irreducible over $\mathrm{GF}(2)$.

We note that a construction analogous to Eq. (3.3) works for every finite field (e.g., a finite field of any prime cardinality), but the problem of providing an explicit representation of such a field remains non-trivial also in other cases (e.g., consider the problem of finding a prime number of size approximately $2^{b(k)}$). The latter fact is the main motivation for considering the following alternative construction for the case of $t = 2$.

The following construction uses (random) affine transformations (as possible seeds). In fact, better performance (i.e., shorter seed length) is obtained by using affine transformations affected by Toeplitz matrices. A Toeplitz matrix is a matrix with all diagonals being homogeneous (see Figure 3.2); that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$ for all i, j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s).

Theorem 3.2 (alternative pairwise independence generator, see Figure 3.2):[5] *Let b, ℓ, ℓ', m: $\mathbb{N}\to\mathbb{N}$ such that $\ell'(k) = \ell(k)/b(k)$ and $m(k) = \lceil \log_2 \ell'(k) \rceil = k - 2b(k) + 1$. Associate $\{0,1\}^n$ with the n-dimensional vector space over $\mathrm{GF}(2)$, and let $v_1, ..., v_{\ell'(k)}$ be fixed distinct vectors in the $m(k)$-dimensional vector space over $\mathrm{GF}(2)$. For $s \in \{0,1\}^{b(k)+m(k)-1}$ and $r \in \{0,1\}^{b(k)}$, let*

$$G(s,r) \stackrel{\mathrm{def}}{=} (T_s v_1 + r\,,\, T_s v_2 + r\,,\, ...,\, T_s v_{\ell'(k)} + r) \tag{3.4}$$

[5] In the common presentation of this pairwise independence generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $2b + \lceil \log_2 \ell' \rceil - 1$.

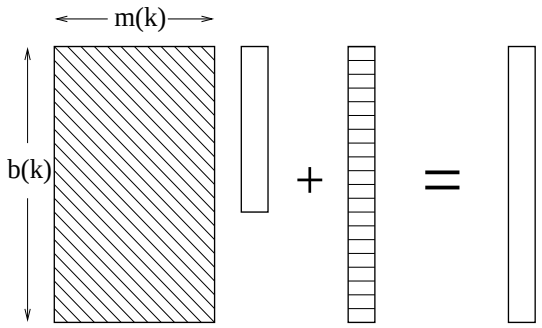

Figure 3.2: An affine transformation affected by a Toeplitz matrix.

where T_s is a $b(k)$-by-$m(k)$ Toeplitz matrix specified by the string s. Then, G is a pairwise independence generator of block-length b and stretch ℓ.

That is, given a seed that represents an affine transformation defined by a $b(k)$-by-$m(k)$ Toeplitz matrix and a $b(k)$-dimensional vector, the generator outputs a sequence of $\ell'(k) \leq 2^{m(k)}$ strings, each of length $b(k)$. Note that $k = 2b(k) + m(k) - 1$, and that the stretching property requires $\ell'(k) > k/b(k)$. We mention that a construction analogous to Eq. (3.4) works for every finite field.

Proof: The current proof is also based on the observation that linear independence of certain expressions yields statistical independence of the corresponding random variables: Here, for every distinct $i, j \in [\ell'(k)]$ and every $\alpha, \beta \in \{0,1\}^{b(k)}$ we consider the $2b(k)$ linear equations over GF(2) that arise from the equalities $\{G(s,r)_i = \alpha\,,\, G(s,r)_j = \beta\}$. This system of $2b(k)$ linear equations over GF(2) refers to Boolean variables representing the bits of s and r, and we shall implicitly show that these equations are linearly-independent.

The argument will refer explicitly to the joint distribution of $G(s,r)_i$ and $G(s,r)_j$ (which equal $T_s v_i + r$ and $T_s v_j + r$, respectively), when $(s,r) \in \{0,1\}^{k-m(k)} \times \{0,1\}^{m(k)}$ is uniformly distributed. (We comment that it is simpler to analyze an analogous construction in which a random matrix is used instead of a Toeplitz matrix; see Exercise 3.4.)[6]

Let us shorthand T_s by T, and refer to selecting at random a uniformly distributed Toeplitz matrix T and a $r \in \mathrm{GF}(2)^b$, where $b = b(k)$ and $m = m(k)$. Then, for every $i \neq j$ and $\alpha, \beta \in \mathrm{GF}(2)^b$, we have

$$\begin{aligned}
\mathsf{Pr}_{T,r}\left[\begin{array}{l} Tv_i + r = \alpha \\ Tv_j + r = \beta \end{array}\right] &= \mathsf{Pr}_{T,r}\left[Tv_i + r = \alpha | Tv_i \oplus Tv_j = \alpha \oplus \beta\right] \cdot \mathsf{Pr}_{T,r}\left[Tv_i \oplus Tv_j = \alpha \oplus \beta\right] \\
&= \mathsf{Pr}_{T,r}\left[Tv_i + r = \alpha | Tw = \gamma\right] \cdot \mathsf{Pr}_{T}\left[Tw = \gamma\right],
\end{aligned}$$

where $w = v_i \oplus v_j \in \mathrm{GF}(2)^m \setminus \{0^m\}$ and $\gamma = \alpha \oplus \beta \in \mathrm{GF}(2)^b$. Clearly, for any $\alpha, \gamma \in \mathrm{GF}(2)^b$ and any b-by-m matrix T' (representing a possible choice of T such that $Tw = \gamma$), it holds

[6] However, using a random matrix requires a longer seed (i.e., $k = (m(k)+1) \cdot b(k)$ rather than $k = 2b(k) + m(k) - 1$), since s should specify an arbitrary $b(k)$-by-$m(k)$ matrix rather than a $b(k)$-by-$m(k)$ Toeplitz matrix.

that:

$$\begin{aligned}\mathsf{Pr}_{T,r}[Tv_i + r = \alpha | Tw = \gamma] &= \mathsf{Pr}_r[T'v_i + r = \alpha] \\ &= 2^{-b}\end{aligned}$$

It is thus left to show that, for any $w \neq 0^m$, when T is a uniformly chosen Toeplitz matrix, the vector Tw is uniformly distributed over $\mathrm{GF}(2)^b$. It may help to consider first the distribution of Mw, where M is a uniformly distributed b-by-m matrix. In this case Mw is merely the sum of several (not zero) uniformly and independently chosen column vectors, and so is uniformly distributed over $\mathrm{GF}(2)^b$. (Indeed, see Exercise 3.4.) The argument regarding a uniformly chosen Toeplitz matrix, which is presented next, is slightly more involved.

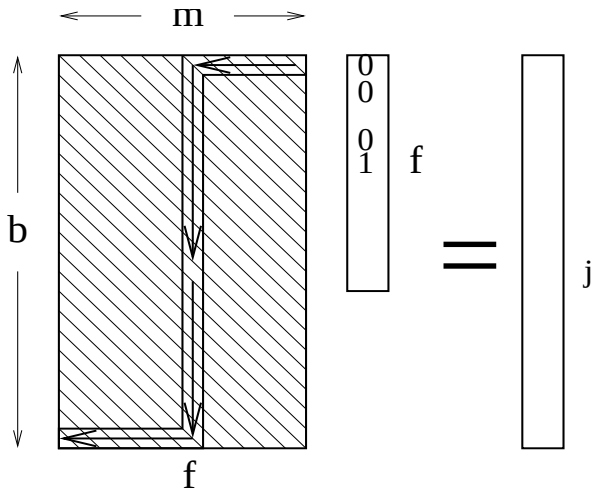

Figure 3.3: The distribution of Tw for a random Toeplitz matrix T.

Let f be the first non-zero entry of $w = (w_1, ..., w_m) \neq 0^m$ (i.e., $w_1 = \cdots = w_{f-1} = 0$ and $w_f = 1$). We make the mental experiment of selecting $T = (t_{i,j})$, by uniformly selecting elements determining T as follows. First we uniformly and independently select $t_{1,m}, ..., t_{1,f}$. Next, we select $t_{2,f}, ..., t_{b,f}$ (here it is important to select $t_{j,f}$ before $t_{j+1,f}$). Finally, we select $t_{b,f-1}, ..., t_{b,1}$. See Figure 3.3. Clearly, this determines a uniformly chosen Toeplitz matrix, denoted T. We conclude by showing that each of the bits of Tw is uniformly distributed given the previous bits. To prove the claim for the j^{th} bit of Tw, consider the time by which $t_{1,m}, ..., t_{1,f}, ..., t_{j-1,f}$ were determined. Note that these determine the first $j-1$ bits of Tw. The key observation is that the value of the j^{th} bit of Tw is a linear combination of the above determined values XORed with the still undetermined $t_{j,f}$. (Here we use the hypothesis that $w_1 = \cdots = w_{f-1} = 0$ and $w_f = 1$.) Thus, uniformly selecting $t_{j,f}$ makes the j^{th} bit of Tw be uniformly distributed given the past. ■

A stronger notion of efficient generation. Ignoring the issue of finding a representation for a large finite field, both the foregoing constructions are efficient in the sense that the generator's output can be produced in time that is polynomial in its length. Actually,

the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. Specifically, there exists a polynomial-time algorithm that given a seed, $s \in \{0,1\}^k$, and a block location $i \in [\ell'(k)]$ (in binary), outputs the i^{th} block of the corresponding output (i.e., the i^{th} block of $G(s)$). Note that, in the case of the first construction (captured by Eq. (3.3)), this stronger notion depends on the ability to find a representation of $\mathrm{GF}(2^{b(k)})$ in $\mathrm{poly}(k)$-time.[7] Recall that this is possible in the case that $b(k)$ is of the form $2 \cdot 3^e$.

3.2.2 A taste of the applications

Pairwise independence generators do suffice for a variety of applications (cf., [113]). Many of these applications are based on the fact that "Laws of Large Numbers" hold for sequences of trials that are pairwise independent (rather than totally independent).[8] This fact stems from the application of Chebyshev's Inequality (cf. Section 1.2.2), and is the basis of the (rather generic) application to ("pairwise independent") sampling (cf. Section 6.4).

As a concrete example, we mention the derandomization of a fast parallel algorithm for the Maximal Independent Set problem (as presented in [83, Sec. 12.3]).[9]

In general, whenever the analysis of a randomized algorithm only relies on the hypothesis that some objects are distributed in a pairwise independent manner, we may replace its random choices by a sequence of choices that is generated by a pairwise independence generator. Thus, pairwise independence generators suffice for fooling distinguishers that are derived from some natural and interesting randomized algorithms.

Referring to Eq. (3.3), we remark that, for any constant $t \geq 2$, the cost of derandomization (i.e., going over all 2^k possible seeds) is exponential in the block-length (because $b(k) = k/t$). On the other hand, the number of blocks is at most exponential in the block-length (because $\ell'(k) \leq 2^{b(k)}$), and so if a larger number of blocks is needed, then we can artificially increase the block-length in order to accommodate this (i.e., set $b(k) = \log_2 \ell'(k)$). Thus, the cost of derandomization is polynomial in $\max(\ell'(k), 2^{b'(k)})$, where $\ell'(k)$ denotes the desired number of blocks and $b'(k)$ the desired block-length. (In other words, $\ell'(k)$ denotes the desired number of random choices, and $2^{b'(k)}$ represents the size of the domain of each of these choices.) It follows that *whenever the analysis of a randomized algorithm can be based on a constant amount of independence* between feasibly-many random choices, each taken within a domain of feasible size, *then a feasible derandomization is possible*.

3.3 Small-Bias Generators

As stated in Section 3.2.2, $O(1)$-wise independence generators allow for the efficient derandomization of any efficient randomized algorithm the analysis of which is only based on a *constant amount of independence* between the bits of its random-tape. This restriction is due to the fact that t-wise independence generators of stretch ℓ require a seed of length

[7] For the basic notion of efficiency, it suffices to find a representation of $\mathrm{GF}(2^{b(k)})$ in $\mathrm{poly}(\ell(k))$-time, which can be done by an exhaustive search in the case that $b(k) = O(\log \ell(k))$.

[8] See discussions in Sections 1.2.4 and 6.4.

[9] The core of this algorithm is picking each vertex with probability that is inversely proportional to the vertex's degree. The analysis only requires that these choices be pairwise independent. Furthermore, these choices can be (approximately) implemented by uniformly selecting values in a sufficiently large set.

$\Omega(t \cdot \log \ell)$. Trying to go beyond constant-independence in such derandomizations (while using seeds of length that is logarithmic in the length of the pseudorandom sequence) was the original motivation of the notion of small-bias generators. Specifically, as we shall see in Section 3.3.2, small-bias generators yield meaningful approximations of t-wise independence sequences (based on logarithmic-length seeds).

While the aforementioned type of derandomizations remains an important application of small-bias generators, the latter are of independent interest and have found numerous other applications. In particular, small-bias generators fool "global tests" that examine the entire output sequence and not merely a fixed number of positions in it (as in the case of limited independence generators). Specifically, a small-bias generator produces a sequence of bits that fools any linear test (i.e., a test that computes a fixed linear combination of the bits).

For $\varepsilon : \mathbb{N} \to [0,1]$, an ε-bias generator with stretch function ℓ is a relatively efficient deterministic algorithm (e.g., working in $\mathrm{poly}(\ell(k))$-time) that expands a k-bit long random seed into a sequence of $\ell(k)$ bits such that for any fixed non-empty set of indices $I \subseteq \{1, ..., \ell(k)\}$ the bias of the output sequence over I is at most $\varepsilon(k)$. The bias of a sequence of n (possibly dependent) Boolean random variables $\zeta_1, ..., \zeta_n \in \{0,1\}$ over a set $I \subseteq \{1, ..., n\}$ is defined as

$$2 \cdot \left| \mathsf{Pr}\left[\bigoplus_{i \in I} \zeta_i = 1\right] - \frac{1}{2} \right| \;=\; \left| \mathsf{Pr}\left[\bigoplus_{i \in I} \zeta_i = 1\right] - \mathsf{Pr}\left[\bigoplus_{i \in I} \zeta_i = 0\right] \right| . \tag{3.5}$$

The factor of 2 was introduced to make these biases correspond to the Fourier coefficients of the distribution (viewed as a function from $\{0,1\}^n$ to the reals). To see the correspondence replace $\{0,1\}$ by $\{\pm 1\}$, and substitute XOR by multiplication. The bias with respect to a set I is thus written as

$$\left| \mathsf{Pr}\left[\prod_{i \in I} \zeta_i = +1\right] - \mathsf{Pr}\left[\prod_{i \in I} \zeta_i = -1\right] \right| \;=\; \left| \mathsf{E}\left[\prod_{i \in I} \zeta_i\right] \right| , \tag{3.6}$$

which is merely the (absolute value of the) Fourier coefficient corresponding to I.

3.3.1 Constructions

Relatively efficient small-bias generators with exponential stretch and exponentially vanishing bias are known.

Theorem 3.3 (small-bias generators):[10] *For some universal constant $c > 0$, let $\ell : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$ such that $\ell(k) \leq \varepsilon(k) \cdot \exp(k/c)$. Then, there exists an ε-bias generator with stretch function ℓ operating in time that is polynomial in the length of its output.*

In particular, we may have $\ell(k) = \exp(k/2c)$ and $\varepsilon(k) = \exp(-k/2c)$. Four simple constructions of small-bias generators that satisfy Theorem 3.3 are known (see [8] and [105,

[10] In the common presentation of this generator, the length of the seed is determined as a function of the desired bias and stretch. That is, given the parameters ε and ℓ, the seed length is set to $c \cdot \log(\ell/\varepsilon)$. We comment that using [8] the constant c is merely 2 (i.e., $k \approx 2\log_2(\ell/\varepsilon)$), whereas using [85] $k \approx \log_2 \ell + 4\log_2(1/\varepsilon)$.

Sec. 3.4]). One of these constructions is based on Linear Feedback Shift Registers (LFSRs), where the seed of the generator is used to determine both the "feedback rule" and the "start sequence" of the LFSR. Specifically, a feedback rule of a t-long LFSR is an irreducible polynomial of degree t over GF(2), denoted $f(z) = z^t + \sum_{j=0}^{t-1} f_j z^j$ where $f_0 = 1$, and the (ℓ-bit long) sequence produced by the corresponding LFSR based on the start sequence $s_0 s_1 \cdots s_{t-1} \in \{0,1\}^t$ is defined as $r_0 r_1 \cdots r_{\ell-1}$, where

$$r_i = \begin{cases} s_i & \text{if } i \in \{0, 1, ..., t-1\}, \\ \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j} & \text{if } i \in \{t, t+1, ..., \ell-1\} \end{cases} \tag{3.7}$$

(see Figure 3.4). As stated previously, in the corresponding small-bias generator the k-bit long seed is used for selecting an *almost* uniformly distributed feedback rule f (i.e., a random irreducible polynomial of degree $t = k/2$) and a uniformly distributed start sequence s (i.e., a random t-bit string).[11] The corresponding $\ell(k)$-bit long output $r = r_0 r_1 \cdots r_{\ell(k)-1}$ is computed as in Eq. (3.7).

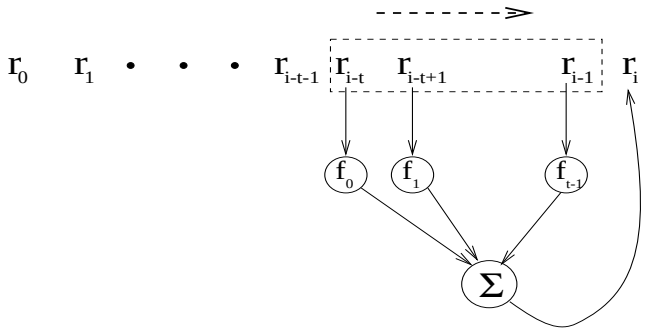

Figure 3.4: The LFSR small-bias generator (for $t = k/2$).

Proposition 3.4 (analysis of the LFSR small-bias generator): *The foregoing generator, captured in Eq.* (3.7) *and depicted in Figure 3.4, has bias at most* $O(\ell/2^t)$.

Proof Sketch: The pivot of the analysis is a correspondance between formal polynomials (in z) reduced modulo the polynomial f and linear combinations of the bits $r_0, r_1, ..., r_{\ell(k)-1}$. This correspondance emerges from the fact that r_i relates to $r_{i-t}, ..., r_{i-1}$ in the same way that z^i relates to $z^{i-t}, ..., z^{i-1}$ (or z^t relates to $z^0, ..., z^{t-1}$). Specifically, we rely on the following observations:

1. Each formal polynomial p of degree at most $\ell(k) - 1$, which is a linear combination of the formal polynomials $z^0, z^1, ..., z^{\ell(k)-1}$. More importantly, the reduction of p modulo the polynomial f can be written as a linear combination of the formal polynomials $z^0, z^1, ..., z^{t-1}$, where the coefficients are functions of f.

[11] Note that an implementation of this generator requires an algorithm for selecting an almost random irreducible polynomial of degree $t = \Omega(k)$. A simple algorithm proceeds by enumerating all irreducible polynomials of degree t, and selecting one of them at random. This algorithm can be implemented (using t random bits) in $\exp(t)$-time, which is $\text{poly}(\ell(k))$ if $\ell(k) = \exp(\Omega(k))$. A $\text{poly}(t)$-time algorithm that uses $O(t)$ random bits is described in [8, Sec. 8].

2. Each linear combination of the bits $r_0, r_1, ..., r_{\ell(k)-1}$ can be written as a linear combination of the bits $s_0, s_1, ..., s_{t-1}$, where the coefficients are functions of f.

3. Furthermore, the coefficients used for presenting the (reduction of the) formal polynomial $p(z) = \sum_{i=0}^{\ell(k)-1} p_i z^i$ (modulo p) equals the coefficients used for presenting (the linear combination) $\sum_{i=0}^{\ell(k)-1} p_i r_i$ as a linear combination of the bits $s_0, s_1, ..., s_{t-1}$.

Indeed, it suffices to prove the foregoing claim for the special case in which a single p_i is 1. Thus, we first prove that r_i equals $\sum_{j=0}^{t-1} c_j^{(f,i)} \cdot s_j$ such that $c_j^{(f,i)}$ is the coefficient of z^j in the (degree $t-1$) polynomial obtained by reducing z^i modulo the polynomial $f(z)$ (i.e., $z^i \equiv \sum_{j=0}^{t-1} c_j^{(f,i)} z^j \pmod{f(z)}$). This is proved by recalling that $z^t \equiv \sum_{j=0}^{t-1} f_j z^j \pmod{f(z)}$, and thus for every $i \geq t$ it holds that $z^i \equiv \sum_{j=0}^{t-1} f_j z^{i-t+j} \pmod{f(z)}$. (Note the correspondence to $r_i = \sum_{j=0}^{t-1} f_j \cdot r_{i-t+j}$.)

Next, for any non-empty index set $I \subseteq \{0, ..., \ell(k)-1\}$, we evaluate the bias of the sequence $r_0, ..., r_{\ell(k)-1}$ over I, where f is a random irreducible polynomial of degree t and $s = (s_0, ..., s_{t-1}) \in \{0,1\}^t$ is uniformly distributed. We consider two cases depending on whether or not $f(z)$ divides $p(z) \stackrel{\text{def}}{=} \sum_{i \in I} z^i$, which is an event that only depends on the random choice of f. The crucial observation is that if f does not divide p, then $\sum_{i \in I} r_i$ is uniformly distributed in $\{0,1\}$ (when the distribution is defined based on a random choice of s). On the other hand, f is unlikely to divide p. Details follow.

- We first claim that, *for a fixed f and random $s \in \{0,1\}^t$, the random variable $\sum_{i \in I} r_i$ has non-zero bias if and only if $f(z)$ divides the polynomial $p(z) = \sum_{i \in I} z^i$.*

  The claim holds because (by the aforementioned correspondance) $\sum_{i \in I} r_i = \sum_{j=0}^{t-1} \sum_{i \in I} c_j^{(f,i)} s_j$, whereas $\sum_{i \in I} z^i = \sum_{j=0}^{t-1} \sum_{i \in I} c_j^{(f,i)} z^j$. Thus, $\sum_{i \in I} r_i$ is a non-zero combination of the s_j's if and only if $\sum_{i \in I} c_j^{(f,i)} \neq 0$ for some j, which means that $f(z)$ does not divide $\sum_{i \in I} z^i$. But whenever $\sum_{i \in I} r_i$ is a non-zero combination of the s_j's, it is the case that $\sum_{i \in I} r_i$ is uniformly distributed in $\{0,1\}$, which means that it has zero bias.

- Next, we observe that *the probability that a random irreducible polynomial of degree t divides a fixed polynomial of degree at most $\ell(k)$ is $\Theta(\ell(k)/2^t)$.*

  This holds because a polynomial of degree n can be divided by at most n/d different irreducible polynomials of degree d. On the other hand, the number of irreducible polynomials of degree d over GF(2) is $\Theta(2^d/d)$.

It follows that for random f and s, the sequence $r_0, ..., r_{\ell(k)-1}$ has bias $O(\ell/2^t)$. □

A stronger notion of efficient generation. As in Section 3.2.1, we note that the aforementioned constructions satisfy a stronger notion of efficient generation, which is useful in several applications. That is, there exists a polynomial-time algorithm that given a k-bit long seed and a bit location $i \in [\ell(k)]$ (in binary), outputs the i^{th} bit of the corresponding output. Specifically, in case of the LFSR construction, given a seed $f_0, ..., f_{(k/2)-1}, s_0, ..., s_{(k/2)-1}$ and a bit location $i \in [\ell(k)]$ (in binary), the algorithm outputs the i^{th} bit of the corresponding

output (i.e., r_i). The assertion is based on the fact that

$$\begin{pmatrix} r_{i-t+1} \\ r_{i-t+2} \\ \vdots \\ r_{i-1} \\ r_i \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ f_0 & f_1 & f_2 & \cdots & f_{t-1} \end{pmatrix} \begin{pmatrix} r_{i-t} \\ r_{i-t+1} \\ \vdots \\ r_{i-2} \\ r_{i-1} \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ f_0 & f_1 & f_2 & \cdots & f_{t-1} \end{pmatrix}^{i-t+1} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{t-2} \\ s_{t-1} \end{pmatrix}$$

3.3.2 A taste of the applications

An archetypical application of small-bias generators is for producing short and random "fingerprints" (or "digests") of strings such that equality and inequality among strings is (probabilistically) reflected in equality and inequality between their corresponding fingerprints. The key observation is that checking whether or not $x = y$ is probabilistically reducible to checking whether the inner product modulo 2 of x and r equals the inner product modulo 2 of y and r, *where r is produced by a small-bias generator G*. Thus, the pair (s, v), where s is a random seed to G and v equals the inner product modulo 2 of z and $G(s)$, serves as the randomized fingerprint of the string z. One advantage of this reduction is that only a few bits (i.e., the seed of the generator and the result of the inner product) need to be "communicated between x and y" in order to enable the checking (see Exercise 3.7). A related advantage is the low randomness complexity of this reduction, which uses $|s|$ rather than $|G(s)|$ random bits, where $|s|$ may be $O(\log |G(s)|)$. This low (i.e., logarithmic) randomness-complexity underlies the application of small-bias generators to the construction of PCP systems and amplifying reductions of gap problems regarding the satisfiability of systems of equations (see, e.g., Exercise 3.17).

Small-bias generators have been used in a variety of areas (e.g., inapproximation, structural complexity, and applied cryptography; see the references in [41, Sec. 3.6.2]). In addition, as shown next, small-bias generators seem an important tool in the design of various types of "pseudorandom" objects.

Approximate independence generators. As hinted at the beginning of this section, small-bias is related to approximate versions of limited independence.[12] Actually, as implied by Exercise 3.8, even a restricted type of ε-bias (in which only subsets of size $t(k)$ are required to have bias upper-bounded by ε) implies that any $t(k)$ bits in the said sequence are $2^{t(k)/2} \cdot \varepsilon(k)$-close to $U_{t(k)}$, where here we refer to the variation distance (i.e., L1-Norm distance) between the two distributions. (The max-norm of the difference is bounded by

[12] We warn that, unlike in the case of perfect independence, here we refer only to the distribution on fixed bit locations. See Exercise 3.6 for further discussion.

$\varepsilon(k)$.)[13] Combining Theorem 3.3 and the foregoing upper-bound, we obtain *generators with exponential stretch* (i.e., $\ell(k) = \exp(\Omega(k))$) that produce *sequences that are approximately $\Omega(k)$-wise independent in the sense that any $t(k) = \Omega(k)$ bits in them are $2^{-\Omega(k)}$-close to* $U_{t(k)}$. Thus, whenever the analysis of a randomized algorithm can be based on a logarithmic amount of (almost) independence between feasibly-many binary random choices, a feasible derandomization is possible (by using an adequate generator of logarithmic seed length).[14]

Extensions to non-binary choices were considered in various works (see references in [41, Sec. 3.6.2]). Some of these works also consider the related problem of constructing small "discrepancy sets" for geometric and combinatorial rectangles.

t-universal set generators. Using the aforementioned upper-bound on the max-norm (of the deviation from uniform of any t locations), any ε-bias generator yields a *t-universal set generator*, provided that $\varepsilon < 2^{-t}$. The latter generator outputs sequences such that in every subsequence of length t all possible 2^t patterns occur (i.e., each for at least one possible seed). Such generators have many applications.

3.3.3 Generalization

In this section, we outline a generalization of the treatment of small-bias generators to the generation of sequences over an arbitrary finite field. Focusing on the case of a field of prime cardinality, denoted $\mathrm{GF}(p)$, we first define an adequate notion of bias (as done in Section 1.3.2.5). Generalizing Eq. (3.6), we define the bias of a sequence of n (possibly dependent) random variables $\zeta_1, ..., \zeta_n \in \mathrm{GF}(p)$ with respect to the linear combination $(c_1, ..., c_n) \in \mathrm{GF}(p)^n$ as $\left\| \mathsf{E}\left[\omega^{\sum_{i=1}^n c_i\zeta_i}\right] \right\|$, where ω denotes the p^{th} (complex) root of unity (i.e., $\omega = -1$ if $p = 2$). We mention that upper-bounds on the biases of $\zeta_1, ..., \zeta_n$ (with respect to any non-zero linear combinations) yield upper-bounds on the distance of $\sum_{i=1}^n c_i\zeta_i$ from the uniform distribution over $\mathrm{GF}(p)$.

We say that $S \subseteq \mathrm{GF}(p)^n$ is an ε-bias probability space if a uniformly selected sequence in S has bias at most ε with respect to any non-zero linear combination over $\mathrm{GF}(p)$. (Whenever such a space is efficiently constructible, it yields a corresponding ε-biased generator.) We mention that the LFSR construction, outlined in Section 3.3.1 and analyzed in Proposition 3.4, generalizes to $\mathrm{GF}(p)$ and yields an ε-bias probability space of size (at most) p^{2e}, where $e = \lceil \log_p(n/\varepsilon) \rceil$. Such constructions can be used in applications that generalize those in Section 3.3.2.

A different generalization. Recalling that small-bias generators fool all linear tests, we consider generators that fool any test that can be represented by a polynomial of degree d.

[13] Both bounds are derived from the L2-Norm bound on the difference vector (i.e., the difference between the two probability vectors). For details, see Exercise 3.8.

[14] Furthermore, as shown in Exercise 3.14, relying on the linearity of the construction presented in Theorem 3.1, we can obtain *generators with double-exponential stretch* (i.e., $\ell(k) = \exp(2^{\Omega(k)})$) *that are approximately $t(k)$-independent* (in the foregoing sense). That is, we may obtain generators with stretch $\ell(k) = 2^{2^{\Omega(k)}}$ producing bit sequences in which any $t(k) = \Omega(k)$ positions have variation distance at most $\varepsilon(k) = 2^{-\Omega(k)}$ from uniform; in other words, such generators may have seed-length $k = O(t(k) + \log(1/\varepsilon(k)) + \log\log\ell(k))$. In the corresponding result for the max-norm distance, it suffices to have $k = O(\log(t(k)/\varepsilon(k)) + \log\log\ell(k))$.

It was recently proved that taking the sum of d independently distributed outputs produced by a small-bias generator (on d independently chosen seeds) yields a sequence that fools all degree d tests [112]. (Interestingly, this sequence may not fool all polynomials of degree $d+1$; see [105].)

3.4 Random Walks on Expanders

In this section we review generators that produce a sequence of values by taking a random walk on a large graph that has a small degree but an adequate "mixing" property (in the sense that a random walk of logarithmic length that starts at any fixed vertex reaches an almost uniformly distributed vertex). Such a graph is called an expander, and by taking a random walk (of length ℓ') on it we generate a sequence of ℓ' values over its vertex set, while using a random seed of length $b+(\ell'-1)\cdot\log_2 d$, where 2^b denotes the number of vertices in the graph and d denotes its degree. This seed length should be compared against the $\ell'\cdot b$ random bits required for generating a sequence of ℓ' independent samples from $\{0,1\}^b$ (or taking a random walk on a clique of size 2^b). Interestingly, as we shall see, the pseudorandom sequence (generated by the said random walk on an expander) *behaves similarly to a truly random sequence with respect to hitting any dense subset of* $\{0,1\}^b$. Let us start by defining this property (or rather by defining the corresponding hitting problem).

Definition 3.5 (the hitting problem): *A sequence of* (possibly dependent) *random variables, denoted* $(X_1,...,X_{\ell'})$*, over* $\{0,1\}^b$ *is* (ε,δ)-hitting *if for any* (target) *set* $T\subseteq\{0,1\}^b$ *of cardinality at least* $\varepsilon\cdot 2^b$*, with probability at least* $1-\delta$*, at least one of these variables hits* T*; that is,* $\Pr[\exists i \text{ s.t. } X_i\in T]\geq 1-\delta$.

Clearly, a truly random sequence of length ℓ' over $\{0,1\}^b$ is (ε,δ)-hitting for $\delta=(1-\varepsilon)^{\ell'}$. The aforementioned "expander random walk generator" (to be described next) achieves similar behavior.[15] Specifically, for arbitrary small $c>0$ (which depends on the degree and the mixing property of the expander), the generator's output is (ε,δ)-hitting for $\delta=(1-(1-c)\cdot\varepsilon)^{\ell'}$. To describe this generator, we need to discuss expanders.

3.4.1 Background: expanders and random walks on them

The current subsection is quite minimal; a more elaborate discussion of expander graphs, their properties and their constructions appears in Lecture 5.

By expander graphs (or expanders) of degree d and eigenvalue bound $\lambda<d$, we actually mean an infinite family of d-regular[16], graphs, $\{G_N\}_{N\in\mathbb{S}}$ ($\mathbb{S}\subseteq\mathbb{N}$), such that G_N is a d-regular graph over N vertices and the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of G_N is upper-bounded by λ. For simplicity, we shall assume that the vertex set of G_N is $[N]$ (although in some constructions a somewhat more redundant representation is more convenient). We will refer to such a family as a (d,λ)-expander (for $\mathbb{S}$). This technical definition is related to the aforementioned notion of "mixing" (which refers

[15] We comment that other pseudorandom generators that were considered in this text also exhibit hitting properties; see Exercise 3.16.

[16] A graph is called d-regular if each of its vertices has exactly d neighbors.

to the rate at which a random walk starting at a fixed vertex reaches uniform distribution over the graph's vertices).

We are interested in explicit constructions of such graphs, by which we mean that there exists a polynomial-time algorithm that on input N (in binary), a vertex v in G_N and an index $i \in \{1,...,d\}$, returns the i^{th} neighbor of v. (We also require that the set $\mathbb{S}$ for which G_N's exist is sufficiently "tractable" – say, that given any $n \in \mathbb{N}$ one may efficiently find an $s \in \mathbb{S}$ such that $n \leq s < 2n$.) Several explicit constructions of expanders are known (cf., e.g., [81, 79, 91]). Below, we rely on the fact that for every $\overline{\lambda} > 0$, there exist d and an explicit construction of a $(d, \overline{\lambda} \cdot d)$-expander over $\{2^b : b \in \mathbb{N}\}$.[17] The relevant (to us) fact about expanders is stated next.

Theorem 3.6 (Expander Random Walk Theorem): *Let $G = (V, E)$ be an expander graph of degree d and eigenvalue bound λ. Consider taking a random walk on G by uniformly selecting a start vertex and taking $\ell' - 1$ additional random steps such that at each step the walk uniformly selects an edge incident at the current vertex and traverses it. Then, for any $W \subseteq V$ and $\rho \stackrel{\text{def}}{=} |W|/|V|$, the probability that such a random walk stays in W is at most*

$$\rho \cdot \left(\rho + (1-\rho) \cdot \frac{\lambda}{d}\right)^{\ell'-1} \quad . \tag{3.8}$$

Thus, a random walk on an expander is "pseudorandom" with respect to the hitting property (i.e., when we consider hitting the set $V \setminus W$ and use $\varepsilon = 1-\rho$); that is, a set of density ε is hit with probability at least $1-\delta$, where $\delta = (1-\varepsilon) \cdot (1-\varepsilon+(\lambda/d) \cdot \varepsilon)^{\ell'-1} < (1-(1-(\lambda/d)) \cdot \varepsilon)^{\ell'}$. A proof of Theorem 3.6 is given in [65], while a proof of an upper-bound that is weaker than Eq. (3.8) is outlined next.

A version of the Expander Random Walk Theorem: Using notation as in Theorem 3.6, we claim that the probability that a random walk of length ℓ' stays in W is at most $(\rho + (\lambda/d)^2)^{\ell'/2}$. In fact, we make a more general claim that refers to the probability that a random walk of length ℓ' intersects $W_0 \times W_1 \times \cdots \times W_{\ell'-1}$. The claimed upper-bound is

$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell'-1} \sqrt{\rho_i + (\lambda/d)^2}, \tag{3.9}$$

where $\rho_i \stackrel{\text{def}}{=} |W_i|/|V|$. In order to prove Eq. (3.9), we view the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in the graph and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first transformation shrinks the component that is orthogonal to the uniform distribution, whereas the second transformation shrinks the component that is in the direction of the uniform distribution. For further details, see Exercise 3.18.

[17] This can be obtained with $d = \text{poly}(1/\overline{\lambda})$. In fact, $d = O(1/\overline{\lambda}^2)$, which is optimal, can be obtained too, albeit with graphs of sizes that are only approximately powers of two.

3.4.2 The generator

Using Theorem 3.6 and an explicit $(2^t, \overline{\lambda} \cdot 2^t)$-expander, we obtain a generator that produces sequences that are (ε, δ)-hitting for δ that is almost optimal.

Proposition 3.7 (The Expander Random Walk Generator):[18] *For every constant $\overline{\lambda} > 0$, consider an explicit construction of $(2^t, \overline{\lambda} \cdot 2^t)$-expanders for $\{2^n : n \in \mathbb{N}\}$, where $t \in \mathbb{N}$ is a sufficiently large constant. For $v \in [2^n] \equiv \{0,1\}^n$ and $i \in [2^t] \equiv \{0,1\}^t$, denote by $\Gamma_i(v)$ the vertex of the corresponding 2^n-vertex graph that is reached from vertex v when following its i^{th} edge. For $b, \ell' : \mathbb{N} \to \mathbb{N}$ such that $k = b(k) + (\ell'(k) - 1) \cdot t < \ell'(k) \cdot b(k)$, and for $v_0 \in \{0,1\}^{b(k)}$ and $i_1, ..., i_{\ell'(k)-1} \in [2^t]$, let*

$$G(v_0, i_1, ..., i_{\ell'(k)-1}) \stackrel{\text{def}}{=} (v_0, v_1, ..., v_{\ell'(k)-1}), \tag{3.10}$$

where $v_j = \Gamma_{i_j}(v_{j-1})$. Then, G has stretch $\ell(k) = \ell'(k) \cdot b(k)$, and $G(U_k)$ is (ε, δ)-hitting for any $\varepsilon > 0$ and $\delta = (1 - (1 - \overline{\lambda}) \cdot \varepsilon)^{\ell'(k)}$.

The stretch of G is maximized at $b(k) \approx k/2$ (and $\ell'(k) = k/2t$), but maximizing the stretch is not necessarily the goal in all applications. In many applications, the parameters n, ε and δ are given, and the goal is to derive a generator that produces (ε, δ)-hitting sequences over $\{0,1\}^n$ while minimizing both the length of the sequence and the amount of randomness used by the generator (i.e., the seed length). Indeed, Proposition 3.7 suggests using sequences of length $\ell' \approx \varepsilon^{-1} \log_2(1/\delta)$ that are generated based on a random seed of length $n + O(\ell')$.

Expander random-walk generators have been used in a variety of areas (e.g., PCP and inapproximability (see [19, Sec. 11.1]), cryptography (see [42, Sec. 2.6]), and the design of various types of "pseudorandom" objects.

Notes

The various generators presented in Lecture 3 were not inspired by any of the other types of pseudorandom generator (nor even by the generic notion of pseudorandomness). Pairwise independence generators were explicitly suggested in [30] (and are implicit in [28]). The generalization to t-wise independence (for $t \geq 2$) is due to [6]. Small-bias generators were first defined and constructed by Naor and Naor [85], and three simple constructions were subsequently given in [8]. The Expander Random Walk Generator was suggested by Ajtai, Komlos, and Szemerédi [3], who discovered that random walks on expander graphs provide a good approximation to repeated independent attempts to hit any fixed subset of sufficient density (within the vertex set). The analysis of the hitting property of such walks was subsequently improved, culminating in the bound cited in Theorem 3.6, which is taken from [65, Cor. 6.1].

Exercises

[18] In the common presentation of this generator, the length of the seed is determined as a function of the desired block-length and stretch. That is, given the parameters b and ℓ', the seed length is set to $b + (\ell' - 1) \cdot t$.

Exercise 3.1 Show that the output of any pseudorandom generator is "statistically distinguishable" from the corresponding uniform distribution; that is, show that, for any stretch function ℓ and any generator G of stretch ℓ, the statistical difference between $G(U_k)$ and $U_{\ell(k)}$ is at least $1 - 2^{-(\ell(k)-k)}$.

Exercise 3.2 Show that placing no computational requirements on the generator enables unconditional results regarding "generators" that fool any family of subexponential-size circuits. That is, making no computational assumptions, prove that there exist functions $G : \{0,1\}^* \to \{0,1\}^*$ such that $\{G(U_k)\}_{k\in\mathbb{N}}$ is (strongly) pseudorandom, while $|G(s)| = 2|s|$ for every $s \in \{0,1\}^*$. Furthermore, show that G can be computed in double-exponential time.

Guideline: Use the Probabilistic Method (cf. Lecture 2). First, for any fixed circuit $C : \{0,1\}^n \to \{0,1\}$, upper-bound the probability that for a random set $S \subset \{0,1\}^n$ of size $2^{n/2}$ the absolute value of $\mathsf{Pr}[C(U_n) = 1] - (|\{x \in S : C(x) = 1\}|/|S|)$ is larger than $2^{-n/8}$. Next, using a union bound, prove the existence of a set $S \subset \{0,1\}^n$ of size $2^{n/2}$ such that no circuit of size $2^{n/5}$ can distinguish a uniformly distributed element of S from a uniformly distributed element of $\{0,1\}^n$, where distinguishing means with a probability gap of at least $2^{-n/8}$.

Exercise 3.3 (adaptive t-wise independence tests) Recall that a generator $G : \{0,1\}^k \to \{0,1\}^{\ell'(k)\cdot b(k)}$ is called t-wise independent if *for any t fixed block positions*, the distribution $G(U_k)$ restricted to these t blocks is uniform over $\{0,1\}^{t\cdot b(k)}$. Prove that the output of a t-wise independence generator is (perfectly) indistinguishable from the uniform distribution *by any test that examines t of the blocks, even if the examined blocks are selected adaptively* (i.e., the location of the i^{th} block to be examined is determined based on the contents of the previously inspected blocks).

Guideline: First show that, without loss of generality, it suffices to consider deterministic (adaptive) testers. Next, show that the probability that such a tester sees any fixed sequence of t values at the locations selected *adaptively* (in the generator's output) equals $2^{-t\cdot b(k)}$, where $b(k)$ is the block-length.

Exercise 3.4 (another pairwise independence generator) Consider a construction analogous to the one in Theorem 3.2, except that here the seed specifies an arbitrary affine $b(k)$-by-$m(k)$ transformation. That is, for $s \in \{0,1\}^{b(k)\cdot m(k)}$ and $r \in \{0,1\}^{b(k)}$, where $k = b(k)\cdot m(k) + b(k)$, let

$$G(s,r) \stackrel{\text{def}}{=} (A_s v_1 + r\,,\, A_s v_2 + r\,,...,\, A_s v_{\ell'(k)} + r) \tag{3.11}$$

where A_s is a $b(k)$-by-$m(k)$ matrix specified by the string s. Show that G as in Eq. (3.11) is a pairwise independence generator of block-length b and stretch ℓ.

Guideline: First note that for every fixed $i \in [\ell'(k)]$, the i^{th} element in the sequence $G(U_k)$, denoted $G(U_k)_i$, is uniformly distributed in $\{0,1\}^{b(k)}$. Actually, show that for every fixed $s \in \{0,1\}^{k-b(k)}$, it holds that $G(s, U_{b(k)})_i$ is uniformly distributed in $\{0,1\}^{b(k)}$. Next note that it suffices to show that, for every $j \neq i$, conditioned on the value of $G(U_k)_i$, the value of $G(U_k)_j$ is uniformly distributed in $\{0,1\}^{b(k)}$. The key technical detail is showing that, for any non-zero vector $v \in \{0,1\}^{m(k)}$ and a uniformly selected $s \in \{0,1\}^{k-b(k)}$, it holds that $A_s v$ is uniformly distributed in $\{0,1\}^{b(k)}$. Let

$i \in [m(k)]$ denote a non-zero coordinate in v (i.e., $v_i \neq 0$). Then, consider any fixing of the columns of A_s other than the i^{th} column and note that when the i^{th} column is selected uniformly the value of $A_s v$ is uniformly distributed in $\{0,1\}^{b(k)}$.

Exercise 3.5 (yet another pairwise independence generator) In continuation of Exercise 3.4, consider the following construction (which appears in the proof of [44, Thm. 7.7]). For $t > 1$, let $b(k) = k/t$, and consider the mapping of $(s^1, ..., s^t) \in \{0,1\}^{t \cdot b(k)}$ to $(r^J) \in \{0,1\}^{(2^t-1) \cdot b(k)}$, where the J's range over all *non-empty* subsets of $\{1, 2, ..., t\}$ and $r^J \stackrel{\text{def}}{=} \bigoplus_{j \in J} s^j$. Prove that G is a pairwise independence generator of block-length b and stretch $\ell(k) = \frac{2^t-1}{t} \cdot k$.

Guideline: For $J \neq J'$, it holds that $r^J \oplus r^{J'} = \bigoplus_{j \in K} s^j$, where K denotes the symmetric difference of J and J'.

Exercise 3.6 (adaptive t-wise independence tests, revisited) Prove that, in contrast to Exercise 3.3, with respect to *non-perfect* indistinguishability, there is a discrepancy between adaptive and non-adaptive tests that inspect t locations.

1. Specifically, present a distribution over 2^{t-1}-bit long strings in which every t fixed bit positions are $t \cdot 2^{-t}$-close to uniform, but there exists a test that adaptively inspects t positions and distinguishes this distribution from the uniform one with gap of $1/2$.

   Guideline: Modify the uniform distribution over $((t-1) + 2^{t-1})$-bit long strings such that the first $t-1$ locations indicate a bit position (among the rest) that is set to zero.

2. On the other hand, prove that if every t fixed bit positions in a distribution X are ε-close to uniform, then every test that adaptively inspects t positions can distinguish X from the uniform distribution with gap at most $2^t \cdot \varepsilon$.

   Guideline: See Exercise 3.3.

Exercise 3.7 Suppose that G is an ε-bias generator with stretch ℓ. Show that equality between the $\ell(k)$-bit strings x and y can be probabilistically checked (with error probability $(1+\varepsilon)/2$) by comparing the inner product modulo 2 of x and $G(s)$ to the inner product modulo 2 of y and $G(s)$, where $s \in \{0,1\}^k$ is selected uniformly. Note that this method is a randomness-efficient approximation of comparing the inner product modulo 2 of x and r to the inner product modulo 2 of y and r, where $r \in \{0,1\}^{\ell(k)}$ is selected uniformly.
(Hint: Consider the special case in which $y = 0^{\ell(k)}$.)

Exercise 3.8 (bias vs. statistical difference from uniform) Let X be a random variable assuming values in $\{0,1\}^t$. Prove that if X has bias at most ε over any non-empty set then the statistical difference between X and U_t is at most $2^{t/2} \cdot \varepsilon$, and that for every $x \in \{0,1\}^t$ it holds that $\mathsf{Pr}[X = x] = 2^{-t} \pm \varepsilon$.

Guideline: See Section 1.3.2.

Exercise 3.9 (a non-trivial example of a small bias probability space) Let S be the set of all n-bit strings having a number of 1-entries that is divisible by three. Then, the uniform distribution over S yields a distribution that has bias $2^{-\Omega(n)}$.[19]

[19] Note that we do not suggest to present a small-bias generator based on this fact: Implementing such a generator is non-trivial, and the expansion obtained is quite poor. Still, the claim itself seems interesting, and was indeed used in [45].

Guideline: The key observation is that the sum modulo 3 of a uniformly distributed n-bit string is almost uniformly distributed in $\{0,1,2\}$, where the error term vanishes exponentially with n. Now, for any non-empty $I \subseteq [n]$, consider the probability that for uniformly distributed string $\sigma_1 \cdots \sigma_n \in \{0,1\}^n$ both $\sum_{i=1}^n \sigma_i \equiv 0 \pmod 3$ and $\sum_{i \in I}^n \sigma_i \equiv 0 \pmod 2$ hold. Show that this probability is very close to $1/6$, where the error term that vanishes expoinentially in $\max(|I|, n-|I|)$. Indeed, consider the two corresponding cases (i.e., $|I| \leq n/2$ and $|I| \geq n/2$). For example, for $|I| \leq n/2$, consider the distributions $\sum_{i \in [n] \setminus I} \sigma_i \bmod 3$ and $\sum_{i \in I}^n \sigma_i \bmod 2$. The other case (i.e., $|I| \geq n/2$) is more interesting: Consider the distributions $\sum_{i \in I} \sigma_i \bmod 3$ and $\sum_{i \in I}^n \sigma_i \bmod 2$, or just the distribution $\sum_{i \in I} \sigma_i \bmod 6$. To analyze the latter distribution consider a $|I|$-step random walk on a directed k-cycle (where $k = 6$) such that an edge is traversed with probability $1/2$ (and the walk remain in place otherwise). It can be shown that the probability that an m-step walk ends at the start vertex is $\frac{1}{k} \pm 2^{-\Omega(m/k^2)}$. For details, see [45, Apdx. A.1].

Exercise 3.10 (on the existence of (non-explicit) small-bias generators)
Prove that, for $k = \log_2(\ell(k)/\varepsilon(k)^2) + O(1)$, there exists a function $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ such that $G(U_k)$ has bias at most $\varepsilon(k)$ over any non-empty subset of $[\ell(k)]$.

Guideline: Use the Probabilistic Method as in Exercise 3.2.

Exercise 3.11 (limitations on small-bias generators) Let G be an ε-bias generator with stretch ℓ, and view G as a mapping from $\mathrm{GF}(2)^k$ to $\mathrm{GF}(2)^{\ell(k)}$. As such, each bit in the output of G can be viewed as a polynomial[20] in the k input variables (each ranging in $\mathrm{GF}(2)$). Prove that if $\varepsilon(k) < 1$ and each of these polynomials has *total degree* at most d, then $\ell(k) \leq \sum_{i=1}^d \binom{k}{i}$. Derive the following corollaries:

1. If $\varepsilon(k) < 1$, then $\ell(k) < 2^k$ (regardless of d).[21]
2. If $\varepsilon(k) < 1$ and $\ell(k) > k$, then G cannot be a linear transformation.[22]

Guideline (for the main claim): Note that, without loss of generality, all the aforementioned polynomials have a free term equal to zero (and have individual degree at most 1 in each variable). Next, consider the vector space spanned by all d-monomials over k variables (i.e., monomials having at most d variables). Since $\varepsilon(k) < 1$, the polynomials representing the output bits of G must correspond to a sequence of independent vectors in this space.

Exercise 3.12 (advanced topic: a sanity check for space-bounded pseudorandomness)
The following fact is suggested as a sanity check for candidate pseudorandom generators with respect to space-bounded automata. The fact (to be proven as an exercise) is that, for every $\varepsilon(\cdot)$ and $s(\cdot)$ such that $s(k) \geq 1$ for every k, if G is (s, ε)-pseudorandom (as per [44, Def. 8.20]), then G is an ε-bias generator.

---

[20] Recall that every Boolean function over $\mathrm{GF}(p)$ can be expressed as a polynomial of *individual degree* at most $p-1$.

[21] This upper-bound is optimal, because (efficient) ε-bias generators of stretch $\ell(k) = \mathrm{poly}(\varepsilon(k)) \cdot 2^k$ do exist (see [85]).

[22] In contrast, bilinear ε-bias generators (i.e., with $\ell(k) > k$) do exist; for example, $G(s) = (s, b(s))$, where $b(s_1, ..., s_k) = \sum_{i=1}^{k/2} s_i s_{(k/2)+i} \bmod 2$, is an ε-bias generator with $\varepsilon(k) = \exp(-\Omega(k))$. (Hint: Focusing on bias over sets that include the last output bit, prove that, without loss of generality, it suffices to analyze the bias of $b(U_k)$.)

Exercise 3.13 In contrast to Exercise 3.12, prove that there exist $\exp(-\Omega(n))$-bias distributions over $\{0,1\}^n$ that are not $(2, 0.666)$-pseudorandom.

Guideline: Show that the uniform distribution over the set

$$\left\{\sigma_1 \cdots \sigma_n : \sum_{i=1}^{n} \sigma_i \equiv 0 \pmod 3\right\}$$

has bias $\exp(-\Omega(n))$. An alternative construction appears in [105, Sec. 3.5].

Exercise 3.14 (approximate t-wise independence generators (cf. [85])) Combining a small-bias generator as in Theorem 3.3 with the t-wise independence generator of Eq. (3.3), and relying on the linearity of the latter, construct a generator producing ℓ-bit long sequences in which any t positions are at most ε-away from uniform (in variation distance), while using a seed of length $O(t + \log(1/\varepsilon) + \log\log\ell)$. (For max-norm a seed of length $O(\log(t/\varepsilon) + \log\log\ell)$ suffices.)

Guideline: First note that, for any t, ℓ' and $b \geq \log_2 \ell'$, the transformation of Eq. (3.3) can be implemented by a fixed linear (over GF(2)) transformation of a $t \cdot b$-bit seed into an ℓ-bit long sequence, where $\ell = \ell' \cdot b$. It follows that, for $b = \log_2 \ell'$, there exists a fixed GF(2)-linear transformation T of a random seed of length $t \cdot b$ into a t-wise independent bit sequence of the length ℓ (i.e., $T\,U_{t \cdot b}$ is t-wise independent over $\{0,1\}^\ell$). Thus, every t rows of T are linearly independent. The key observation is that when we replace the aforementioned random seed by an ε'-bias sequence, every set of $i \leq t$ positions in the output sequence has bias at most ε' (because they define a non-zero linear test on the bits of the ε'-bias sequence). Note that the length of the new seed (used to produce ε'-bias sequence of length $t \cdot b$) is $O(\log tb/\varepsilon')$. Applying Exercise 3.8, we conclude that any t positions are at most $2^{t/2} \cdot \varepsilon'$-away from uniform (in variation distance). Recall that this was obtained using a seed of length $O(\log(t/\varepsilon') + \log\log\ell)$, and the claim follows by using $\varepsilon' = 2^{-t/2} \cdot \varepsilon$.

Exercise 3.15 (small-bias generator and error-correcting codes) Show a correspondence between ε-bias generators of stretch ℓ and binary linear error-correcting codes mapping $\ell(k)$-bit long strings to 2^k-bit long strings such that every two codewords are at distance $(1 \pm \varepsilon(k)) \cdot 2^{k-1}$ apart.

Guideline: Associate $\{0,1\}^k$ with $[2^k]$. Then, a generator $G : [2^k] \to \{0,1\}^{\ell(k)}$ corresponds to the code $C : \{0,1\}^{\ell(k)} \to \{0,1\}^{2^k}$ such that, for every $i \in [\ell(k)]$ and $j \in [2^k]$, the i^{th} bit of $G(j)$ equals the j^{th} bit of $C(0^{i-1}10^{\ell(k)-i})$.

Exercise 3.16 (advanced topic: other pseudorandom generators and the hitting problem) Show that various pseudorandom generators yield solutions to the hitting problem (as defined in Definition 3.5). Specifically:

1. Show that a pairwise independence generator of block-length b and stretch ℓ yields a sequence over $\{0,1\}^b$ that is (ε, δ)-hitting for $\delta = O(1/\varepsilon\ell')$, where $\ell' = \ell/b$.

   Advanced exercise: Show that when using t-wise independence. the error bound can be reduced to $\delta = O(t^2/\varepsilon\ell')^{\lfloor t/2 \rfloor}$.

2. Referring to [44, Def. 8.20], show that a (b, δ)-pseudorandom generator of stretch ℓ yields a sequence over $\{0,1\}^b$ that is (ε, δ)-hitting for $\delta = (1-\varepsilon)^{\ell/b} + \delta$.

3. Consider modifications of the hitting problem in which the target set T is restricted to be recognizable within some specified complexity.

   (a) Show that a general-purpose pseudorandom generator of stretch ℓ yields a sequence over $\{0,1\}^b$ that is (ε,δ)-hitting for target sets in $\mathcal{BPP}$ and $\delta = (1-\varepsilon)^{\ell/b} + 1/p$, where p is an arbitrary polynomial.

   (b) Referring to [44, Def. 8.14], show that a canonical derandomizer of stretch ℓ yields a sequence over $\{0,1\}^b$ that is (ε,δ)-hitting for target sets that are recognized by circuits of size ℓ^2 and $\delta = (1-\varepsilon)^{\ell/b} + 1/6$.

What is the advantage of using the expander random walk generator over each of the foregoing options?

Exercise 3.17 (natural inapproximability without the PCP Theorem) In contrast to the inapproximability results that are based on the PCP Theorem (cf., e.g., [44, §10.1.1.2]), the NP-completeness of the following gap problem can be established (rather easily) without referring to the PCP Theorem. The instances of this problem are systems of quadratic equations over GF(2) (as in [44, Exer. 2.25]), yes-instances are systems that have a solution, and no-instances are systems for which any assignment violates at least one third of the equations.

Guideline: As stated in [44, Exer. 2.25], when given such a quadratic system, it is NP-hard to determine whether or not there exists an assignment that satisfies all the equations. Using an adequate small-bias generator (cf. Section 3.3), present an amplifying reduction (cf. [44, Sec. 9.3.3]) of the foregoing problem to itself (i.e., yes-instances are mapped to yes-instances, whereas systems that have no solution are mapped to systems for which any assignment violates at least one third of the equations). Specifically, if the input system has m equations, then we use a generator that defines a sample space of poly(m) many m-bit strings, and consider the corresponding linear combinations of the input equations. Note that it suffices to bound the bias of the generator by 1/6, whereas using an ε-biased generator yields an analogous result with 1/3 replaced by $0.5-\varepsilon$.

Exercise 3.18 (a version of the Expander Random Walk Theorem) Let $G=(V,E)$ be a graph as in Theorem 3.6. Prove that the probability that a random walk of length ℓ' intersects $W_0 \times W_1 \times \cdots \times W_{\ell'-1} \subseteq V^{\ell'}$ is upper bounded by Eq. (3.9).

Guideline: Let A be a matrix representing the random walk on G (i.e., A is the adjacency matrix of G divided by d), and let $\hat{\lambda} \stackrel{\text{def}}{=} \lambda/d$. Note that the uniform distribution, represented by the vector $\overline{u} = (N^{-1},...,N^{-1})^\top$, is the eigenvector of A that is associated with the largest eigenvalue (which is 1), whereas all other eigenvalues have absolute value at most $\hat{\lambda}$. Let P_i be a 0-1 matrix that has 1-entries only on its diagonal such that entry (j,j) is set to 1 if and only if $j \in W_i$. Then, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is the sum of the entries of the vector $\overline{v} \stackrel{\text{def}}{=} P_{\ell-1}A \cdots P_2 A P_1 A P_0 \overline{u}$. We are interested in upper-bounding $\|\overline{v}\|_1$, and use $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$, where $\|\overline{z}\|_1$ and $\|\overline{z}\|$ denote the L_1-norm and L_2-norm of $\overline{z}$, respectively (e.g., $\|\overline{u}\|_1 = 1$ and $\|\overline{u}\| = N^{-1/2}$). The key observation is that the linear transformation P_iA shrinks every vector. For further details, see proof of Lemma 5.5.

Exercise 3.19 Using notation as in Theorem 3.6, prove that the probability that a random walk of length ℓ' visits W more than $\alpha\ell'$ times is smaller than $\binom{\ell'}{\alpha\ell'} \cdot (\rho + (\lambda/d)^2)^{\alpha\ell'/2}$. For

example, for $\alpha = 1/2$ and $\lambda/d < \sqrt{\rho}$, we get an upper-bound of $(32\rho)^{\ell'/4}$. We comment that much better bounds can be obtained (cf., e.g., [58]).

Guideline: Use a union bound on all possible sequences of $m = \alpha\ell'$ visits, and upper-bound the probability of visiting W in steps $j_1, ..., j_m$ by applying Eq. (3.9) with $W_i = W$ if $i \in \{j_1, ..., j_m\}$ and $W = V$ otherwise.

Lecture 4

Hashing

Hashing is extensively used in complexity theory in order to map arbitrary (unstructured) sets "almost uniformly" to a smaller structured set of adequate size. Specifically, hashing is supposed to map an arbitrary 2^m-subset (of $\{0,1\}^n$) to $\{0,1\}^m$ in an "almost uniform" manner.

For a fixed set S of cardinality 2^m, a 1-1 mapping $f_S : S \to \{0,1\}^m$ does exist, but it is not necessarily an efficient one (e.g., it may require "knowing" the entire set S). Clearly, no fixed function $f : \{0,1\}^n \to \{0,1\}^m$ can map every 2^m subset of $\{0,1\}^n$ to $\{0,1\}^m$ in a 1-1 manner (or even approximately so). However, a random function $f : \{0,1\}^n \to \{0,1\}^m$ has the property that, for every 2^m-subset $S \subset \{0,1\}^n$, with overwhelmingly high probability f maps S to $\{0,1\}^m$ such that no point in the range has many f-preimages in S. The problem is that a truly random function is unlikely to have a succinct representation (let alone an efficient evaluation algorithm). We seek families of functions that have a similar property, but do have a succinct representation as well as an efficient evaluation algorithm.

4.1 Definitions

Motivated by the foregoing discussion, we consider families of functions $\{H_n^m\}_{m<n}$ such that the following properties hold:

1. For every $S \subset \{0,1\}^n$, with high probability, a function h selected uniformly in H_n^m maps S to $\{0,1\}^m$ in an "almost uniform" manner. For example, for any $|S| = 2^m$ and each point y, with high probability over the choice of h, it holds that $|\{x \in S : h(x) = y\}| \leq \text{poly}(n)$.

2. The functions in H_n^m have succinct representation. For example, we may require that $H_n^m \equiv \{0,1\}^{\ell(n,m)}$, for some polynomial ℓ.

3. The functions in H_n^m can be efficiently evaluated. That is, there exists a polynomial-time algorithm that, on input a representation of a function, h (in H_n^m), and a string $x \in \{0,1\}^n$, returns $h(x)$. In some cases we make even more stringent requirements regarding the the algorithm (e.g., that it runs in linear space).

Condition 1 was left vague on purpose. At the very least, we require that (for every $y \in \{0,1\}^m$) the expected size of $\{x \in S : h(x) = y\}$ equals $|S|/2^m$. We shall see (in Section 4.3) that different (stronger) interpretations of Condition 1 are satisfied by different types of hashing functions. We focus on t-wise independent hashing functions, defined next.

Definition 4.1 (t-wise independent hashing functions): *A family H_n^m of functions from n-bit strings to m-bit strings is called t-*wise independent *if for every t distinct domain elements $x_1, ..., x_t \in \{0,1\}^n$ and every $y_1, ..., y_t \in \{0,1\}^m$ it holds that*

$$\mathsf{Pr}_{h \in H_n^m}[\wedge_{i=1}^t h(x_i) = y_1] \;=\; 2^{-t \cdot m}$$

That is, every t domain elements are mapped by a uniformly chosen $h \in H_n^m$ in a totally uniform manner. Note that for $t \geq 2$, it follows that the probability that a random $h \in H_n^m$ maps two distinct domain elements to the same image is 2^{-m}. Such (families of) functions are called universal (cf. [28]), but we will focus on the stronger condition of t-wise independence.

4.2 Constructions

The following constructions are merely a re-interpretation of the constructions presented in the context of (special purpose) pseudorandom generators; specifically, see Section 3.2.1. (Alternatively, one may view the latter constructions as a re-interpretation of the following two constructions.)

Construction 4.2 (t-wise independent hashing): *For $t, m, n \in \mathbb{N}$ such that $m \leq n$, consider the following family of hashing functions mapping n-bit strings to m-bit strings. Each t-sequence $\overline{s} = (s_0, s_1, ..., s_{t-1}) \in \{0,1\}^{t \cdot n}$ describes a function $h_{\overline{s}} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{\overline{s}}(x)$ equals the m-bit prefix of the binary representation of $\sum_{j=0}^{t-1} s_j x^j$, where the arithmetic is that of $\mathrm{GF}(2^n)$, the finite field of 2^n elements.*

Proposition 3.1 implies that Construction 4.2 constitutes a family of t-wise independent hash functions. Typically, we will use either $t = 2$ or $t = \Theta(n)$. To make the construction totally explicit, we need an explicit representation of $\mathrm{GF}(2^n)$. An alternative construction for the case of $t = 2$ may be obtained as follows. Recall that a Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j.

Construction 4.3 (Alternative pairwise independent hashing): *For $m \leq n$, consider the family of hashing functions in which each n-by-m Toeplitz matrix T and an m-dimensional vector b describes a function $h_{T,b} : \{0,1\}^n \to \{0,1\}^m$ such that $h_{T,b}(x) = Tx + b$.*

Proposition 3.2 implies that Construction 4.3 constitutes a family of pairwise independent hash functions. Note that a n-by-m Toeplitz matrix can be specified by $n + m - 1$ bits, yielding description length $n + 2m - 1$. An alternative construction (using $m \cdot n + m$ bits of representation) uses arbitrary n-by-m matrices rather than Toeplitz matrices.

4.3 The Leftover Hash Lemma

We now turn to the "almost uniform" cover condition (i.e., Condition 1) mentioned in Section 4.1. One concrete interpretation of this condition is implied by the following lemma.

Lemma 4.4 *Let $m < n$ be integers, H_n^m be a family of pairwise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $y \in \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{\varepsilon^2|S|}$ fraction of $h \in H_n^m$ it holds that*

$$|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot \frac{|S|}{2^m} \tag{4.1}$$

By pairwise independence (or rather even by "1-wise independence"), the expected size of $\{x \in S : h(x) = y\}$ is $|S|/2^m$, where the expectation is taken uniformly over all $h \in H_n^m$. The lemma upper bounds the fraction of h's that deviate from the expected value. Needless to say, the bound is meaningful only in case $|S| > 2^m$ (or alternatively for $\varepsilon > 1$). Setting $\varepsilon = \sqrt[3]{2^m/|S|}$ (and focusing on the case that $|S| > 2^m$), we infer that *for all but at most an ε fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$. Thus, each range element has approximately the right number of h-preimages in the set S under almost all $h \in H_n^m$.

Proof: Fixing an arbitrary set $S \subseteq \{0,1\}^n$ and an arbitrary $y \in \{0,1\}^m$, we estimate the probability that a uniformly selected $h \in H_n^m$ violates Eq. (4.1). We define random variables ζ_x, over the aforementioned probability space, such that $\zeta_x = \zeta_x(h)$ equal 1 if $h(x) = y$ and 0 otherwise. The expected value of $\sum_{x \in S} \zeta_x$ is $\mu \stackrel{\text{def}}{=} |S| \cdot 2^{-m}$, and we are interested in the probability that this sum deviates from the expectation. Applying Chebyshev's Inequality, we get

$$\mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| > \varepsilon \cdot \mu\right] \;<\; \frac{\mu}{\varepsilon^2 \mu^2}$$

because $\mathsf{Var}(\sum_{x \in S} \zeta_x) < |S| \cdot 2^{-m}$ by the pairwise independence of the ζ_x's and the fact that $\mathsf{E}[\zeta_x] = 2^{-m}$. The lemma follows. ■

A generalization (called mixing). The proof of Lemma 4.4 can be easily extended to show that *for every set $T \subset \{0,1\}^m$ and every $\varepsilon > 0$, for all but at most an $\frac{2^m}{|T| \cdot |S| \varepsilon^2}$ fraction of $h \in H_n^m$ it holds that* $|\{x \in S : h(x) \in T\}| = (1 \pm \varepsilon) \cdot |T| \cdot |S|/2^m$; see Exercise 4.1. In the case that $m = n$, this is called a mixing property, and is meaningfull provided $|T| \cdot |S| > 2^m/\varepsilon$.

An extremely useful corollary. The aforementioned generalization of Lemma 4.4 asserts that most functions behave well with respect to any fixed sets of preimages $S \subset \{0,1\}^n$ and images $T \subset \{0,1\}^m$. A seemingly stronger statement, which is (non-trivially) implied by Lemma 4.4, is that for all adequate sets S most functions $h \in H_n^m$ map S to $\{0,1\}^m$ in an almost uniform manner.[1] This is a consequence of the following theorem.

[1] That is, for X as in Theorem 4.5 and any $\alpha > 0$, for all but at most an α fraction of the functions $h \in H_n^m$ it holds that $h(X)$ is $(2\varepsilon/\alpha)$-close to U_m.

Theorem 4.5 (a.k.a Leftover Hash Lemma): *Let H_n^m and $S \subseteq \{0,1\}^n$ be as in Lemma 4.4, and define $\varepsilon = \sqrt[3]{2^m/|S|}$. Consider random variable X and H that are uniformly distributed on S and H_n^m, respectively. Then, the statistical distance between $(H, H(X))$ and (H, U_m) is at most 2ε.*

Using the terminology of Section 8, we say that H_n^m yields a strong extractor (with parameters to be spelled out there).

Proof: Let V denote the set of pairs (h,y) that violate Eq. (4.1), and $\overline{V} \stackrel{\text{def}}{=} (H_n^m \times \{0,1\}^m) \setminus V$. Then for every $(h,y) \in \overline{V}$ it holds that

$$\begin{aligned} \mathsf{Pr}[(H,H(X)) = (h,y)] &= \mathsf{Pr}[H=h] \cdot \mathsf{Pr}[h(X)=y] \\ &= (1 \pm \varepsilon) \cdot \mathsf{Pr}[(H,U_m) = (h,y)]. \end{aligned}$$

On the other hand, by Lemma 4.4 (which asserts $\mathsf{Pr}[(H,y) \in V] \leq \varepsilon$ for every $y \in \{0,1\}^m$), we have $\mathsf{Pr}[(H,U_m) \in V] \leq \varepsilon$. Using

$$\begin{aligned} \mathsf{Pr}[(H,H(X)) \in V] &= 1 - \mathsf{Pr}[(H,H(X)) \in \overline{V}] \\ &\leq 1 - \mathsf{Pr}[(H,U_n)) \in \overline{V}] + \varepsilon \;\leq\; 2\varepsilon \end{aligned}$$

we upper-bounded the statistical difference between $(H,H(X))$ and (H,U_m) by

$$\begin{aligned} &\frac{1}{2} \cdot \sum_{(h,y) \in H_n^m \times \{0,1\}^m} |\mathsf{Pr}[(H,H(X)) = (h,y)] - \mathsf{Pr}[(H,U_m) = (h,y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} |\mathsf{Pr}[(H,H(X)) = (h,y)] - \mathsf{Pr}[(H,U_m) = (h,y)]| \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot \sum_{(h,y) \in V} (\mathsf{Pr}[(H,H(X)) = (h,y)] + \mathsf{Pr}[(H,U_m) = (h,y)]) \\ &\leq \frac{\varepsilon}{2} + \frac{1}{2} \cdot (2\varepsilon + \varepsilon) \end{aligned}$$

and the claim follows. ■

An alternative proof of Theorem 4.5. Define the collision probability of a random variable Z, denote $\mathsf{cp}(Z)$, as the probability that two independent samples of Z yield the same result. Alternatively, $\mathsf{cp}(Z) \stackrel{\text{def}}{=} \sum_z \mathsf{Pr}[Z=z]^2$. Theorem 4.5 follows by combining the following two facts:

1. A general fact: *If $Z \in [N]$ and $\mathsf{cp}(Z) \leq (1+4\epsilon^2)/N$ then Z is ϵ-close to the uniform distribution on $[N]$.*

   We prove the contra-positive: Assuming that the statistical distance between Z and the uniform distribution on $[N]$ equals δ, we show that $\mathsf{cp}(Z) \geq (1+4\delta^2)/N$. This is done by defining $L \stackrel{\text{def}}{=} \{z : \mathsf{Pr}[Z=z] < 1/N\}$, and lower-bounding $\mathsf{cp}(Z)$ by using the fact that the collision probability minimizes on uniform distributions. Specifically,

$$\mathsf{cp}(Z) \geq |L| \cdot \left(\frac{\mathsf{Pr}[Z \in L]}{|L|}\right)^2 + (N - |L|) \cdot \left(\frac{\mathsf{Pr}[Z \in [N] \setminus L]}{N - |L|}\right)^2,$$

which equals $1 + (\delta^2/(1-\rho)\rho) \geq 1 + 4\delta^2$, where $\rho = |L|/N$.

2. The collision probability of $(H, H(X))$ is at most $(1 + (2^m/|S|))/(|H_n^m| \cdot 2^m)$. (Furthermore, this holds even if H_n^m is only universal.)

   The proof is by a straightforward calculation. Specifically, note that $\mathsf{cp}(H, H(X)) = |H_n^m|^{-1} \cdot \mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X))]$, whereas $\mathsf{E}_{h \in H_n^m}[\mathsf{cp}(h(X)] = |S|^{-2} \sum_{x_1, x_2 \in S} \mathsf{Pr}[H(x_1) = H(x_2)]$. The sum equals $|S| + (|S|^2 - |S|) \cdot 2^{-m}$, and so $\mathsf{cp}(H, H(X)) < |H_n^m|^{-1} \cdot (2^{-m} + |S|^{-1})$.

Note that it follows that $(H, H(X))$ is $\sqrt{2^m/4|S|}$-close to (H, U_m), which is a stronger bound than the one provided in Theorem 4.5.

Stronger uniformity via higher independence. Recall that Lemma 4.4 asserts that for each point in the range of the hash function, with high probability over the choice of the hash function, *this fixed point* has approximately the expected number of preimages in S. A stronger condition asserts that, with high probability over the choice of the hash function, *every point in its range* has approximately the expected number of preimages in S. Such a guarantee can be obtained when using n-wise independent hashing functions.

Lemma 4.6 *Let $m < n$ be integers, H_n^m be a family of n-wise independent hash functions, and $S \subseteq \{0,1\}^n$. Then, for every $\varepsilon \in (0,1)$, for all but at most an $2^m \cdot (n \cdot 2^m/\varepsilon^2|S|)^{n/2}$ fraction of $h \in H_n^m$, it holds that $|\{x \in S : h(x) = y\}| = (1 \pm \varepsilon) \cdot |S|/2^m$ for every $y \in \{0,1\}^m$.*

Indeed, the lemma should be used with $2^m < \varepsilon^2|S|/4n$. In particular, using $m = \log_2 |S| - \log_2(5n/\varepsilon^2)$ guarantees that with high probability each range elements has $(1 \pm \varepsilon) \cdot |S|/2^m$ preimages in S. Under this setting of parameters $|S|/2^m = 5n/\varepsilon^2$, which is $\mathrm{poly}(n)$ whenever $\varepsilon = 1/\mathrm{poly}(n)$. Needless to say, this guarantee is stronger than the conclusion of Theorem 4.5.

Proof: The proof follows the footsteps of the proof of Lemma 4.4, taking advantage of the fact that the random variables (i.e., the ζ_x's) are now $2t$-wise independent, where $t = n/2$. This allows for the use of a so-called $2t^{\text{th}}$ moment analysis, which generalizes the analysis of pairwise independent samplying (presented in Section 1.2). As in the proof of Lemma 4.4, we fix any S and y, and define $\zeta_x = \zeta_x(h) = 1$ if and only if $h(x) = y$. Letting $\mu = \mathsf{E}[\sum_{x \in S} \zeta_x] = |S|/2^m$ and $\overline{\zeta}_x = \zeta_x - \mathsf{E}(\zeta_x)$, we start with Markov inequality:

$$\begin{aligned} \mathsf{Pr}\left[\left|\mu - \sum_{x \in S} \zeta_x\right| > \varepsilon \cdot \mu\right] &< \frac{\mathsf{E}[(\sum_{x \in S} \overline{\zeta}_x)^{2t}]}{\varepsilon^{2t}\mu^{2t}} \\ &= \frac{\sum_{x_1,\ldots,x_{2t} \in S} \mathsf{E}[\prod_{i=1}^{2t} \overline{\zeta}_{x_i}]}{\varepsilon^{2t} \cdot (|S|/2^m)^{2t}} \end{aligned} \tag{4.2}$$

Using $2t$-wise independence, we note that only the terms in Eq. (4.2) that do not vanish are those in which each variable appears with multiplicity. This mean that only terms having less than t distinct variables contribute to Eq. (4.2). Now, for every $j \leq t$, we have less than

$\binom{|S|}{j} \cdot (2t!) < (2t!/j!) \cdot |S|^j$ terms with j distinct variables, and each contributes less than $(2^{-m})^j$ to the sum. Thus, Eq. (4.2) is upper-bounded by

$$\frac{2t!}{(\varepsilon^{2t}|S|/2^m)^{2t}} \cdot \sum_{j=1}^{t} \frac{(|S|/2^m)^j}{j!} \;<\; \frac{2t!/t!}{(\varepsilon^2|S|/2^m)^t} \;<\; \left(\frac{2t \cdot 2^m}{\varepsilon^2|S|}\right)^t$$

where the first inequality assumes $|S| > n2^m$ (since the claim hold vacuously otherwise). This upper-bounds the probability that a random $h \in H_n^m$ violates the mapping condition regarding a fixed y. Using a union bound on all $y \in \{0,1\}^m$, the lemma follows. ∎

Notes

Pairwise independent hashing functions were essentially introduced by Carter and Wegman [28], who actually considered weaker condition (which they called Universal Hash Functions): Specifically, in contrast to Definition 4.1, they only required that the collision probability is as in the case of random mappings; that is, for every two distinct domain elements $x, y \in \{0,1\}^n$ it holds that $\mathsf{Pr}_{h \in H_n^m}[h(x) = h(y)] = 2^{-m}$.

The Leftover Hash Lemma (i.e., Theorem 4.5) was discovered independently in [23, 61], yet it is an extension of the ideas underlying [96]. The name Leftover Hash Lemma was coined in [62]. A generalized statement of the Theorem 4.5 refers to an arbitrary random variable X over $\{0,1\}^n$ that satisfies $\mathsf{Pr}[X = x] \leq \varepsilon^3 \cdot 2^{-m}$, for every x. This generalization is supported by essentially the same proof. Alternatively, the generalization can be deduced from Theorem 4.5 by noting that such X can be represented by a convex combination of distributions that are each uniform over some set of size $2^m/\varepsilon^3$ (see Exercise 8.1).

Exercises

Exercise 4.1 (a mixing lemma or a generalization of Lemma 4.4) Let $m < n$ and H_n^m be as in Lemma 4.4. Then, for every $S \subseteq \{0,1\}^n$ and $T \subset \{0,1\}^m$ (and every $\varepsilon > 0$), for all but at most an $\frac{2^m}{|T| \cdot |S| \varepsilon^2}$ fraction of $h \in H_n^m$ it holds that $|\{x \in S : h(x) \in T\}| = (1 \pm \varepsilon) \cdot |T| \cdot |S|/2^m$.

Guideline: Just follow the proof of Lemma 4.4, when defining $\zeta_x = \zeta(h) = 1$ if $h(x) \in T$ and 0 otherwise.

Lecture 5

Expander Graphs

In this lecture we review basic facts regarding expander graphs that are most relevant to the current book. For a wider perspective, the interested reader is referred to [60].

Loosely speaking, expander graphs are regular graphs of small degree that exhibit various properties of cliques.[1] In particular, we refer to properties such as the relative sizes of cuts in the graph (i.e., relative to the number of edges), and the rate at which a random walk converges to the uniform distribution (relative to the logarithm of the graph size to the base of its degree).

Some technicalities. Typical presentations of expander graphs refer to one of several variants. For example, in some sources, expanders are presented as bipartite graphs, whereas in others they are presented as ordinary graphs (and are in fact very far from being bipartite). We shall follow the latter convention. Furthermore, at times we implicitly consider an augmentation of these graphs where self-loops are added to each vertex. For simplicity, we also allow parallel edges.

We often talk of expander graphs while we actually mean an infinite collection of graphs such that each graph in this collection satisfies the same property (which is informally attributed to the collection). For example, when talking of a d-regular expander (graph) we actually refer to an infinite collection of graphs such that each of these graphs is d-regular. Typically, such a collection (or family) contains a single N-vertex graph for every $N \in \mathbb{S}$, where $\mathbb{S}$ is an infinite subset of $\mathbb{N}$. Throughout this section, we denote such a collection by $\{G_N\}_{N\in\mathbb{S}}$, with the understanding that G_N is a graph with N vertices and $\mathbb{S}$ is an infinite set of natural numbers.

5.1 Definitions and Properties

We consider two definitions of expander graphs, two different notions of explicit constructions, and two useful properties of expanders.

[1] Another useful intuition is that expander graphs exhibit various properties of random regular graphs of the same degree.

5.1.1 Two mathematical definitions

We start with two different definitions of expander graphs. These definitions are qualitatively equivalent and even quantitatively related. We start with an algebraic definition, which seems technical in nature but is actually the definition typically used in complexity theoretic applications, since it directly implies various "mixing properties" (see Section 5.1.3). We later present a very natural combinatorial definition (which is the source of the term "expander").

The algebraic definition (eigenvalue gap). Identifying graphs with their adjacency matrix, we consider the eigenvalues (and eigenvectors) of a graph (or rather of its adjacency matrix). Any d-regular graph $G = (V, E)$ has the uniform vector as an eigenvector corresponding to the eigenvalue d, and if G is connected and non-bipartite then the absolute values of all other eigenvalues are strictly smaller than d. The eigenvalue bound, denoted $\lambda(G) < d$, of such a graph G is defined as a tight upper-bound on the *absolute value* of all the other eigenvalues. (In fact, in this case it holds that $\lambda(G) < d - \Omega(1/d|V|^2)$.)[2] The algebraic definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* eigenvalue bound that holds for all the graphs in the family.

Definition 5.1 *An infinite family of d-regular graphs, $\{G_N\}_{N\in\mathbb{S}}$, where $\mathbb{S} \subseteq \mathbb{N}$,* satisfies the eigenvalue bound *β if for every $N \in \mathbb{S}$ it holds that $\lambda(G_N) \leq \beta$. In such a case, we say that $\{G_N\}_{N\in\mathbb{S}}$ is a family of (d, β)-*expanders*, and call $d - \beta$ its* eigenvalue gap.

It will be often convenient to consider relative (or normalized) versions of the foregoing quantities, obtained by division by d.

The combinatorial definition (expansion). Loosely speaking, expansion requires that any (not too big) set of vertices of the graph has a relatively large set of neighbors. Specifically, a graph $G = (V, E)$ is c-expanding if, for every set $S \subset V$ of cardinality at most $|V|/2$, it holds that

$$\Gamma_G(S) \stackrel{\text{def}}{=} \{v : \exists u \in S \text{ s.t. } \{u, v\} \in E\} \tag{5.1}$$

has cardinality at least $(1 + c) \cdot |S|$. Assuming the existence of self-loops on all vertices, the foregoing requirement is equivalent to requiring that $|\Gamma_G(S) \setminus S| \geq c \cdot |S|$. In this case, every connected graph $G = (V, E)$ is $(1/|V|)$-expanding.[3] The combinatorial definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* expansion bound that holds for all the graphs in the family.

Definition 5.2 *An infinite family of d-regular graphs, $\{G_N\}_{N\in\mathbb{S}}$* is c-expanding *if for every $N \in \mathbb{S}$ it holds that G_N is c-expanding.*

The two definitions of expander graphs are related (see [9, Sec. 9.2] or [60, Sec. 4.5]). Specifically, the "expansion bound" and the "eigenvalue bound" are related as follows.

[2] This follows from the connection to the combinatorial definition (see Theorem 5.3). Specifically, the square of this graph, denoted G^2, is $|V|^{-1}$-expanding and thus it holds that $\lambda(G)^2 = \lambda(G^2) < d^2 - \Omega(|V|^{-2})$.

[3] In contrast, a bipartite graph $G = (V, E)$ is not expanding, because it always contains a set S of size at most $|V|/2$ such that $|\Gamma_G(S)| \leq |S|$ (although it may hold that $|\Gamma_G(S) \setminus S| \geq |S|$).

Theorem 5.3 *Let G be a d-regular graph having a self-loop on each vertex.*[4]

1. *The graph G is c-expanding for $c \geq (d - \lambda(G))/2d$.*
2. *If G is c-expanding then $d - \lambda(G) \geq c^2/(4 + 2c^2)$.*

Thus, any non-zero bound on the combinatorial expansion of a family of d-regular graphs yields a non-zero bound on its eigenvalue gap, and vice versa. Note, however, that the back-and-forth translation between these measures is not tight. We note that most applications in complexity theory refer to the algebraic definition, and that the loss incurred in Theorem 5.3 is immaterial for them.

Amplification. The "quality of expander graphs improves" by raising these graphs to any power $t > 1$ (i.e., raising their adjacency matrix to the t^{th} power), where this operation corresponds to replacing t-paths (in the original graphs) by edges (in the resulting graphs). Specifically, when considering the algebraic definition, it holds that $\lambda(G^t) = \lambda(G)^t$, but indeed the degree also gets raised to the power t. Still, the ratio $\lambda(G^t)/d^t$ deceases with t. An analogous phenomenon occurs also under the combinatorial definition, provided that some suitable modifications are applied. For example, if for every $S \subseteq V$ it holds that $|\Gamma_G(S)| \geq \min((1+c)\cdot|S|, |V|/2)$, then for every $S \subseteq V$ it holds that $|\Gamma_{G^t}(S)| \geq \min((1+c)^t\cdot|S|, |V|/2)$.

The optimal eigenvalue bound. For every d-regular graph $G = (V, E)$, it holds that $\lambda(G) \geq 2\gamma_G \cdot \sqrt{d-1}$, where $\gamma_G = 1 - O(1/\log_d |V|)$. Thus, for any infinite family of (d, λ)-expanders, it must holds that $\lambda \geq 2\sqrt{d-1}$.

5.1.2 Two levels of explicitness

Towards discussing various notions of explicit constructions of graphs, we need to fix a representation of such graphs. Specifically, throughout this section, when referring to an infinite family of graphs $\{G_N\}_{N\in\mathbb{S}}$, we shall assume that the vertex set of G_N equals $[N]$. Indeed, at times, we shall consider vertex sets having a different structure (e.g., $[m]\times[m]$ for some $m \in \mathbb{N}$), but in all these cases there exists a simple isomorphism of these sets to the canonical representation (i.e., there exists an efficiently computable and invertible mapping of the vertex set of G_N to $[N]$).

A mild notion of explicit constructiveness refers to the *complexity of constructing the entire object* (i.e., the graph).[5] Applying this notion to our setting, we say that an infinite family of graphs $\{G_N\}_{N\in\mathbb{S}}$ is explicitly constructible if there exists a *polynomial-time algorithm that, on input 1^N* (where $N \in \mathbb{S}$), *outputs the list of the edges in the N-vertex graph*

[4] Recall that in such a graph $G = (V, E)$ it holds that $\Gamma_G(S) \supseteq S$ for every $S \subseteq V$, and thus $|\Gamma_G(S)| = |\Gamma_G(S) \setminus S| + |S|$. Furthermore, in such a graph all eigenvalues are greater than or equal to $-d + 1$, and thus if $d - \lambda(G) < 1$ then this is due to a positive eigenvalue of G. These facts are used for bridging the gap between Theorem 5.3 and the more standard versions (see, e.g., [9, Sec. 9.2]) that refer to variants of both definitions. Specifically, [9, Sec. 9.2] refers to $\Gamma^+_G(S) = \Gamma_G(S) \setminus S$ and $\lambda_2(G)$, where $\lambda_2(G)$ is the second largest eigenvalue of G, rather than referring to $\Gamma_G(S)$ and $\lambda(G)$. Note that, in general, $\Gamma_G(S)$ may be attained by the difference between the smallest eigenvalue of G (which may be negative) and $-d$.

[5] Compare Section 2.3.1, where mild and strong notions of explicitness (for error-correcting codes) are discussed.

G_N. That is, the entire graph is constructed in time that is polynomial in its size (i.e., in poly(N)-time).

The foregoing (mild) level of explicitness suffices when the application requires holding the entire graph and/or when the running-time of the application is lower-bounded by the size of the graph. In contrast, other applications refer to a huge virtual graph (which is much bigger than their running time), and only require the computation of the neighborhood relation in such a graph. In this case, the following stronger level of explicitness is relevant.

A strongly explicit construction of an infinite family of (d-regular) graphs $\{G_N\}_{N\in\mathbb{S}}$ is a *polynomial-time algorithm that on input $N \in \mathbb{S}$* (in binary), *a vertex v in the N-vertex graph G_N* (i.e., $v \in [N]$), *and an index $i \in [d]$, returns the i^{th} neighbor of v*. That is, the "neighbor query" is answered in time that is polylogarithmic in the size of the graph. Needless to say, this strong level of explicitness implies the basic (mild) level.

An additional requirement, which is often forgotten but is very important, refers to the "tractability" of the set $\mathbb{S}$. Specifically, we require the existence of an *efficient algorithm that given any $n \in \mathbb{N}$ finds an $s \in \mathbb{S}$ such that $n \leq s < 2n$*. Corresponding to the two foregoing levels of explicitness, "efficient" may mean either running in time poly(n) or running in time poly($\log n$). The requirement that $n \leq s < 2n$ suffices in most applications, but in some cases a smaller interval (e.g., $n \leq s < n + \sqrt{n}$) is required, whereas in other cases a larger interval (e.g., $n \leq s < \text{poly}(n)$) suffices.

Greater flexibility. In continuation to the foregoing paragraph, we comment that expanders can be combined in order to obtain expanders for a wider range of graph sizes. For example, given two d-regular c-expanding graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where $|V_1| \leq |V_2|$ and $c \leq 1$, we can obtain a $(d+1)$-regular $c/2$-expanding graph on $|V_1| + |V_2|$ vertices by connecting the two graphs using a perfect matching of V_1 and $|V_1|$ of the vertices of V_2 (and adding self-loops to the remaining vertices of V_2). More generally, combining the d-regular c-expanding graphs $G_1 = (V_1, E_1)$ through $G_t = (V_t, E_t)$, where $N' \stackrel{\text{def}}{=} \sum_{i=1}^{t-1} |V_i| \leq |V_t|$, yields a $(d+1)$-regular $c/2$-expanding graph on $\sum_{i=1}^{t} |V_i|$ vertices (by using a perfect matching of $\cup_{i=1}^{t-1} V_i$ and N' of the vertices of V_t).

5.1.3 Two properties

The following two properties provide a quantitative interpretation to the statement that expanders approximate the complete graph (or behave approximately like a complete graph). When referring to (d, λ)-expanders, the deviation from the behavior of a complete graph is represented by an error term that is linear in λ/d.

The mixing lemma. Loosely speaking, the following (folklore) lemma asserts that in expander graphs (for which $\lambda \ll d$) the fraction of edges connecting two large sets of vertices approximately equals the product of the densities of these sets. This property is called *mixing*.

Lemma 5.4 (Expander Mixing Lemma): *For every d-regular graph $G = (V, E)$ and for every two subsets $A, B \subseteq V$ it holds that*

$$\left| \frac{|(A \times B) \cap \vec{E}|}{|\vec{E}|} - \frac{|A|}{|V|} \cdot \frac{|B|}{|V|} \right| \leq \frac{\lambda(G)\sqrt{|A| \cdot |B|}}{d \cdot |V|} \leq \frac{\lambda(G)}{d} \tag{5.2}$$

where $\vec{E}$ denotes the set of directed edges (i.e., vertex pairs) *that correspond to the undirected edges of G* (i.e., $\vec{E} = \{(u, v) : \{u, v\} \in E\}$ and $|\vec{E}| = d|V|$).

In particular, $|(A \times A) \cap \vec{E}| = (\rho(A) \cdot d \pm \lambda(G)) \cdot |A|$, where $\rho(A) = |A|/|V|$. It follows that $|(A \times (V \setminus A)) \cap \vec{E}| = ((1 - \rho(A)) \cdot d \pm \lambda(G)) \cdot |A|$.

Proof: Let $N \stackrel{\text{def}}{=} |V|$ and $\lambda \stackrel{\text{def}}{=} \lambda(G)$. For any subset of the vertices $S \subseteq V$, we denote its density in V by $\rho(S) \stackrel{\text{def}}{=} |S|/N$. Hence, Eq. (5.2) is restated as

$$\left| \frac{|(A \times B) \cap \vec{E}|}{d \cdot N} - \rho(A) \cdot \rho(B) \right| \leq \frac{\lambda\sqrt{\rho(A) \cdot \rho(B)}}{d} .$$

We proceed by providing bounds on the value of $|(A \times B) \cap \vec{E}|$. To this end we let $\overline{a}$ denote the N-dimensional Boolean vector having 1 in the i^{th} component if and only if $i \in A$. The vector $\overline{b}$ is defined similarly. Denoting the adjacency matrix of the graph G by $M = (m_{i,j})$, we note that $|(A \times B) \cap \vec{E}|$ equals $\overline{a}^\top M \overline{b}$ (because $(i, j) \in (A \times B) \cap \vec{E}$ if and only if it holds that $i \in A$, $j \in B$ and $m_{i,j} = 1$). We consider the *orthogonal eigenvector basis*, $\overline{e_1}, ..., \overline{e_N}$, where $\overline{e_1} = (1, ..., 1)^\top$ and $\overline{e_i}^\top \overline{e_i} = N$ for each i, and write each vector as a linear combination of the vectors in this basis. Specifically, we denote by a_i the coefficient of $\overline{a}$ in the direction of $\overline{e_i}$; that is, $a_i = (\overline{a}^\top \overline{e_i})/N$ and $\overline{a} = \sum_i a_i \overline{e_i}$. Note that $a_1 = (\overline{a}^\top \overline{e_1})/N = |A|/N = \rho(A)$ and $\sum_{i=1}^N a_i^2 = (\overline{a}^\top \overline{a})/N = |A|/N = \rho(A)$. Similarly for $\overline{b}$. It now follows that

$$\begin{aligned} |(A \times B) \cap \vec{E}| &= \overline{a}^\top M \sum_{i=1}^N b_i \overline{e_i} \\ &= \sum_{i=1}^N b_i \lambda_i \cdot \overline{a}^\top \overline{e_i} \end{aligned}$$

where λ_i denotes the i^{th} eigenvalue of M. Note that $\lambda_1 = d$ and for every $i \geq 2$ it holds that $|\lambda_i| \leq \lambda$. Thus,

$$\begin{aligned} \frac{|(A \times B) \cap \vec{E}|}{dN} &= \sum_{i=1}^N \frac{b_i \lambda_i \cdot a_i}{d} \\ &= \rho(A)\rho(B) + \sum_{i=2}^N \frac{\lambda_i a_i b_i}{d} \\ &\in \left[\rho(A)\rho(B) \pm \frac{\lambda}{d} \cdot \sum_{i=2}^N a_i b_i \right] \end{aligned}$$

Using $\sum_{i=1}^{N} a_i^2 = \rho(A)$ and $\sum_{i=1}^{N} b_i^2 = \rho(B)$, and applying Cauchy-Schwartz Inequality, we bound $\sum_{i=2}^{N} a_i b_i$ by $\sqrt{\rho(A)\rho(B)}$. The lemma follows. ■

The random walk lemma. Loosely speaking, the first part of the following lemma asserts that, as far as remaining "trapped" in some subset of the vertex set is concerned, a random walk on an expander approximates a random walk on the complete graph.

Lemma 5.5 (Expander Random Walk Lemma): *Let $G = ([N], E)$ be a d-regular graph, and consider walks on G that start from a uniformly chosen vertex and take $\ell - 1$ additional random steps, where in each such step we uniformly selects one out of the d edges incident at the current vertex and traverses it.*

Part 1 – Theorem 3.6 (restated): *Let W be a subset of $[N]$ and $\rho \stackrel{\text{def}}{=} |W|/N$. Then the probability that such a random walk stays in W is at most*

$$\rho \cdot \left(\rho + (1-\rho) \cdot \frac{\lambda(G)}{d}\right)^{\ell-1} \tag{5.3}$$

Part 2 – Exercise 3.18 (restated): *For any $W_0, ..., W_{\ell-1} \subseteq [N]$, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is at most*

$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell-1} \sqrt{\rho_i + (\lambda/d)^2}, \tag{5.4}$$

where $\rho_i \stackrel{\text{def}}{=} |W_i|/N$.

The basic principle underlying Lemma 5.5 was discovered by Ajtai, Komlos, and Szemerédi [3], who proved a bound as in Eq. (5.4). The better analysis yielding Theorem 3.6 is due to [65, Cor. 6.1]. A more general bound that refer to the probability of visiting W for a number of times that approximates $|W|/N$ is given in [58], which actually considers an even more general problem (i.e., obtaining Chernoff-type bounds for random variables that are generated by a walk on an expander).

Proof of Equation (5.4): The basic idea is viewing events occuring during the random walk as an evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in G and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first transformation shrinks the component that is orthogonal to the uniform distribution, whereas the second transformation shrinks the component that is in the direction of the uniform distribution. Details follow.

Let A be a matrix representing the random walk on G (i.e., A is the adjacency matrix of G divided by d), and let $\hat{\lambda} \stackrel{\text{def}}{=} \lambda(G)/d$ (i.e., $\hat{\lambda}$ upper-bounds the absolute value of every eigenvalue of A except the first one). Note that the uniform distribution, represented by the vector $\overline{u} = (N^{-1}, ..., N^{-1})^\top$, is the eigenvector of A that is associated with the largest eigenvalue (which is 1). Let P_i be a 0-1 matrix that has 1-entries only on its diagonal such

that entry (j,j) is set to 1 if and only if $j \in W_i$. Then, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is the sum of the entries of the vector

$$\overline{v} \stackrel{\text{def}}{=} P_{\ell-1}A \cdots P_2 A P_1 A P_0 \overline{u}. \tag{5.5}$$

We are interested in upper-bounding $\|\overline{v}\|_1$, and use $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$, where $\|\overline{z}\|_1$ and $\|\overline{z}\|$ denote the L_1-norm and L_2-norm of $\overline{z}$, respectively (e.g., $\|\overline{u}\|_1 = 1$ and $\|\overline{u}\| = N^{-1/2}$). The key observation is that the linear transformation P_iA shrinks every vector.

Main Claim. For every $\overline{z}$, it holds that $\|P_iA\overline{z}\| \leq (\rho_i + \hat{\lambda}^2)^{1/2} \cdot \|\overline{z}\|$.

Proof. Intuitively, A shrinks the component of $\overline{z}$ that is orthogonal to $\overline{u}$, whereas P_i shrinks the component of $\overline{z}$ that is in the direction of $\overline{u}$. Specifically, we decompose $\overline{z} = \overline{z_1} + \overline{z_2}$ such that $\overline{z_1}$ is the projection of $\overline{z}$ on $\overline{u}$ and $\overline{z_2}$ is the component orthogonal to $\overline{u}$. Then, using the triangle inequality and other obvious facts (which imply $\|P_iA\overline{z_1}\| = \|P_i\overline{z_1}\|$ and $\|P_iA\overline{z_2}\| \leq \|A\overline{z_2}\|$), we have

$$\begin{aligned} \|P_iA\overline{z_1} + P_iA\overline{z_2}\| &\leq \|P_iA\overline{z_1}\| + \|P_iA\overline{z_2}\| \\ &\leq \|P_i\overline{z_1}\| + \|A\overline{z_2}\| \\ &\leq \sqrt{\rho_i} \cdot \|\overline{z_1}\| + \hat{\lambda} \cdot \|\overline{z_2}\| \end{aligned}$$

where the last inequality uses the fact that P_i shrinks any uniform vector by eliminating $1-\rho_i$ of its elements, whereas A shrinks the length of any eigenvector except $\overline{u}$ by a factor of at least $\hat{\lambda}$. Using the Cauchy-Schwartz inequality[6], we get

$$\begin{aligned} \|P_iA\overline{z}\| &\leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|\overline{z_1}\|^2 + \|\overline{z_2}\|^2} \\ &= \sqrt{\rho_i + \hat{\lambda}^2} \cdot \|\overline{z}\| \end{aligned}$$

where the equality is due to the fact that $\overline{z_1}$ is orthogonal to $\overline{z_2}$. □

Recalling Eq. (5.5) and using the Main Claim (and $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$), we get

$$\begin{aligned} \|\overline{v}\|_1 &\leq \sqrt{N} \cdot \|P_{\ell-1}A \cdots P_2 A P_1 A P_0 \overline{u}\| \\ &\leq \sqrt{N} \cdot \left(\prod_{i=1}^{\ell-1} \sqrt{\rho_i + \hat{\lambda}^2} \right) \cdot \|P_0\overline{u}\|. \end{aligned}$$

Finally, using $\|P_0\overline{u}\| = \sqrt{\rho_0 N \cdot (1/N)^2} = \sqrt{\rho_0/N}$, we establish Eq. (5.4). ■

Rapid mixing. A property related to Lemma 5.5 is that a random walk starting at any vertex converges to the uniform distribution on the expander vertices after a logarithmic number of steps. Specifically, we claim that *starting at any distribution* $\overline{s}$ (including a distribution that assigns all weight to a single vertex) *after* ℓ *steps on a* (d,λ)*-expander*

[6] That is, we get $\sqrt{\rho_i}\|z_1\| + \hat{\lambda}\|z_2\| \leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|z_1\|^2 + \|z_2\|^2}$, by using $\sum_{i=1}^n a_i \cdot b_i \leq \left(\sum_{i=1}^n {a_i}^2\right)^{1/2} \cdot \left(\sum_{i=1}^n {b_i}^2\right)^{1/2}$, with $n=2$, $a_1 = \sqrt{\rho_i}$, $b_1 = \|z_1\|$, etc.

$G = ([N], E)$ we reach a distribution that is $\sqrt{N}\cdot(\lambda/d)^\ell$-close to the uniform distribution over $[N]$. Using notation as in the proof of Eq. (5.4), the claim asserts that $\|A^\ell\overline{s}-\overline{u}\|_1 \leq \sqrt{N}\cdot\hat{\lambda}^\ell$, which is meaningful only for $\ell > 0.5\cdot\log_{1/\hat{\lambda}} N$. The claim is proved by recalling that $\|A^\ell\overline{s}-\overline{u}\|_1 \leq \sqrt{N}\cdot\|A^\ell\overline{s}-\overline{u}\|$ and using the fact that $\overline{s}-\overline{u}$ is orthogonal to $\overline{u}$ (because the former is a zero-sum vector). Thus, $\|A^\ell\overline{s}-\overline{u}\| = \|A^\ell(\overline{s}-\overline{u})\| \leq \hat{\lambda}^\ell\|\overline{s}-\overline{u}\|$ and using $\|\overline{s}-\overline{u}\| < 1$ the claim follows.

5.2 Constructions

Many explicit constructions of (d,λ)-expanders are known. The first such construction was presented in [81] (where $\lambda < d$ was not explicitly bounded), and an optimal construction (i.e., an optimal eigenvalue bound of $\lambda = 2\sqrt{d-1}$) was first provided in [79]. Most of these constructions are quite simple (see, e.g., Section 5.2.1), but their analysis is based on non-elementary results from various branches of mathematics. In contrast, the construction of Reingold, Vadhan, and Wigderson [91], presented in Section 5.2.2, is based on an iterative process, and its analysis is based on a relatively simple algebraic fact regarding the eigenvalues of matrices.

Before turning to these explicit constructions we note that it is relatively easy to prove the existence of 3-regular expanders, by using the Probabilistic Method (see Lecture 2) and referring to the combinatorial definition of expansion: For details, see Section 2.4.

5.2.1 The Margulis–Gabber–Galil Expander

For every natural number m, consider the graph with vertex set $\mathbb{Z}_m \times \mathbb{Z}_m$ and the edge set in which every $\langle x,y\rangle \in \mathbb{Z}_m\times\mathbb{Z}_m$ is connected to the vertices $\langle x\pm y, y\rangle$, $\langle x\pm(y+1), y\rangle$, $\langle x, y\pm x\rangle$, and $\langle x, y\pm(x+1)\rangle$, where the arithmetic is modulo m. This yields an extremely simple 8-regular graph with an eigenvalue bound that is a constant $\lambda < 8$ (which is independent of m). Thus, we get:

Theorem 5.6 *There exists a strongly explicit construction of a family of $(8, 7.9999)$-expanders for graph sizes $\{m^2 : m\in\mathbb{N}\}$. Furthermore, the neighbors of a vertex in these expanders can be computed in logarithmic-space.*[7]

An appealing property of Theorem 5.6 is that, for every $n \in \mathbb{N}$, it directly yields expanders with vertex set $\{0,1\}^n$. This is obvious in case n is even, but can be easily achieved also for odd n (e.g., use two copies of the graph for $n-1$, and connect the two copies by the obvious perfect matching).

Theorem 5.6 is due to Gabber and Galil [39], building on the basic approach suggested by Margulis [81]. We mention again that the (strongly explicit) (d,λ)-expanders of [79] achieve the optimal eigenvalue bound (i.e., $\lambda = 2\sqrt{d-1}$), but there are annoying restrictions on the

[7] In fact, for m that is a power of two (and under a suitable encoding of the vertices), the neighbors can be computed by a on-line algorithm that uses a constant amount of space. The same holds also for a variant in which each vertex $\langle x,y\rangle$ is connected to the vertices $\langle x\pm 2y, y\rangle$, $\langle x\pm(2y+1), y\rangle$, $\langle x, y\pm 2x\rangle$, and $\langle x, y\pm(2x+1)\rangle$. This variant yields a better known bound on λ, i.e., $\lambda \leq 5\sqrt{2} \approx 7.071$.

degree d (i.e., $d-1$ should be a prime congruent to 1 modulo 4) and on the graph sizes for which this construction works.[8]

5.2.2 The Iterated Zig-Zag Construction

The starting point of the following construction is a very good expander G of *constant size*, which may be found by an exhaustive search. The construction of a large expander graph proceeds in iterations, where in the i^{th} iteration the current graph G_i and the fixed graph G are combined, resulting in a larger graph G_{i+1}. The combination step guarantees that the expansion property of G_{i+1} is at least as good as the expansion of G_i, while G_{i+1} maintains the degree of G_i and is a constant times larger than G_i. The process is initiated with $G_1 = G^2$ and terminates when we obtain a graph G_t of approximately the desired size (which requires a logarithmic number of iterations).

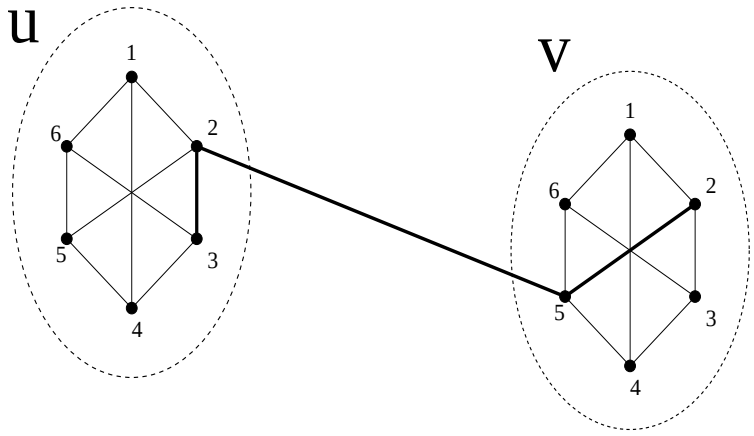

Figure 5.1: Detail of the Zig-Zag product of G' and G. In this example G' is 6-regular and G is a 3-regular graph having six vertices. In the graph G' (not shown), the 2nd edge of vertex u is incident at v, as its 5th edge. The wide 3-segment line shows one of the corresponding edges of $G' ⓩ G$, which connects the vertices $\langle u, 3\rangle$ and $\langle v, 2\rangle$.

The Zig-Zag product. The heart of the combination step is a new type of "graph product" called *Zig-Zag product*. This operation is applicable to any pair of graphs $G = ([D], E)$ and $G' = ([N], E')$, provided that G' (which is typically larger than G) is D-regular. For simplicity, we assume that G is d-regular (where typically $d \ll D$). The Zig-Zag product of G' and G, denoted $G' ⓩ G$, is defined as a graph with vertex set $[N] \times [D]$ and an edge set that includes an edge between $\langle u, i\rangle \in [N] \times [D]$ and $\langle v, j\rangle$ if and only if $\{i, k\}, \{\ell, j\} \in E$ and the k^{th} edge incident at u equals the ℓ^{th} edge incident at v. That is, $\langle u, i\rangle$ and $\langle v, j\rangle$ are connected in $G' ⓩ G$ if there exists a "three step sequence" consisting of a G-step from $\langle u, i\rangle$

[8] The construction in [79] allows graph sizes of the form $(p^3 - p)/2$, where $p \equiv 1 \pmod 4$ is a prime such that $d-1$ is a quadratic residue modulo p. As stated in [7, Sec. 2], the construction can be extended to graph sizes of the form $(p^{3k} - p^{3k-2})/2$, for any $k \in \mathbb{N}$ and p as in the foregoing.

to $\langle u,k\rangle$ (according to the edge $\{i,k\}$ of G), followed by a G'-step from $\langle u,k\rangle$ to $\langle v,\ell\rangle$ (according to the k^{th} edge of u in G' (which is the ℓ^{th} edge of v)), and a final G-step from $\langle v,\ell\rangle$ to $\langle v,j\rangle$ (according to the edge $\{\ell,j\}$ of G). See Figure 5.1 as well as further formalization (which follows).

Teaching note: The following paragraph, which provides a formal description of the zig-zag product, can be ignored in first reading but is useful for more advanced discussion.

It will be convenient to represent graphs like G' by their edge-rotation function, denoted $R' : [N]\times[D] \to [N]\times[D]$, such that $R'(u,i) = (v,j)$ if $\{u,v\}$ is the i^{th} edge incident at u as well as the j^{th} edge incident at v. That is, R' rotates the pair (u,i), which represents one "side" of the edge $\{u,v\}$ (i.e., the side incident at u as its i^{th} edge), resulting in the pair (v,j), which represents the other side of the same edge (which is the j^{th} edge incident at v). For simplicity, we assume that the (constant-size) d-regular graph $G = ([D],E)$ is edge-colorable with d colors, which in turn yields a natural edge-rotation function (i.e., $R(i,\alpha) = (j,\alpha)$ if the edge $\{i,j\}$ is colored α). We will denote by $E_\alpha(i)$ the vertex reached from $i \in [D]$ by following the edge colored α (i.e., $E_\alpha(i) = j$ iff $R(i,\alpha) = (j,\alpha)$). The Zig-Zag product of G' and G, denoted $G'ⓩG$, is then defined as a graph with the vertex set $[N]\times[D]$ and the edge-rotation function

$$(\langle u,i\rangle,\langle\alpha,\beta\rangle) \mapsto (\langle v,j\rangle,\langle\beta,\alpha\rangle) \quad \text{if } R'(u,E_\alpha(i)) = (v,E_\beta(j)). \tag{5.6}$$

That is, edges are labeled by pairs over $[d]$, and the $\langle\alpha,\beta\rangle^{\text{th}}$ edge out of vertex $\langle u,i\rangle \in [N]\times[D]$ is incident at the vertex $\langle v,j\rangle$ (as its $\langle\beta,\alpha\rangle^{\text{th}}$ edge) if $R(u,E_\alpha(i)) = (v,E_\beta(j))$, where indeed $E_\beta(E_\beta(j)) = j$. Intuitively, based on $\langle\alpha,\beta\rangle$, we first take a G-step from $\langle u,i\rangle$ to $\langle u,E_\alpha(i)\rangle$, then viewing $\langle u,E_\alpha(i)\rangle \equiv (u,E_\alpha(i))$ as a side of an edge of G' we rotate it (i.e., we effectively take a G'-step) reaching $(v,j') \stackrel{\text{def}}{=} R'(u,E_\alpha(i))$, and finally we take a G-step from $\langle v,j'\rangle$ to $\langle v,E_\beta(j')\rangle$.

Clearly, the graph $G'ⓩG$ is d^2-regular and has $D\cdot N$ vertices. The key fact, proved in [91] (using techniques as in Section 5.1.3), is that the relative eigenvalue-value of the zig-zag product is upper-bounded by the sum of the relative eigenvalue-values of the two graphs; that is, $\bar\lambda(G'ⓩG) \le \bar\lambda(G') + \bar\lambda(G)$, where $\bar\lambda(\cdot)$ denotes the relative eigenvalue-bound of the relevant graph. The (qualitative) fact that $G'ⓩG$ is an expander if both G' and G are expanders is very intuitive (e.g., consider what happens if G' or G is a clique). Things are even more intuitive if one considers the (related) replacement product of G' and G, denoted $G'ⓡG$, *where there is an edge between* $\langle u,i\rangle \in [N]\times[D]$ *and* $\langle v,j\rangle$ *if and only if either* $u = v$ *and* $\{i,j\} \in E$ *or the* i^{th} *edge incident at* u *equals the* j^{th} *edge incident at* v.

The iterated construction. The iterated expander construction uses the aforementioned zig-zag product as well as graph squaring. Specifically, the construction starts[9] with the d^2-regular graph $G_1 = G^2 = ([D],E^2)$, where $D = d^4$ and $\bar\lambda(G) < 1/4$, and proceeds in iterations such that $G_{i+1} = G_i^2ⓩG$ for $i = 1,2,...,t-1$, where t is logarithmic in the desired graph size. That is, in each iteration, the current graph is first squared and then composed

[9] Recall that, for a sufficiently large constant d, we first find a d-regular graph $G = ([d^4],E)$ satisfying $\bar\lambda(G) < 1/4$, by exhaustive search.

with the fixed (d-regular D-vertex) graph G via the zig-zag product. This process maintains the following two invariants:

1. The graph G_i is d^2-regular and has D^i vertices.

   (The degree bound follows from the fact that a zig-zag product with a d-regular graph always yields a d^2-regular graph.)

2. The relative eigenvalue-bound of G_i is smaller than one half (i.e., $\bar{\lambda}(G_i) < 1/2$).

   (Here we use the fact that $\bar{\lambda}(G_{i-1}^2 ⓩ G) \leq \bar{\lambda}(G_{i-1}^2) + \bar{\lambda}(G)$, which in turn equals $\bar{\lambda}(G_{i-1})^2 + \bar{\lambda}(G) < (1/2)^2 + (1/4)$. Note that graph squaring is used to reduce the relative eigenvalue of G_i before increasing it by zig-zag product with G.)

In order to show that we can actually construct G_i, we show that we can compute the edge-rotation function that correspond to its edge set. This boils down to showing that, given the edge-rotation function of G_{i-1}, we can compute the edge-rotation function of G_{i-1}^2 as well as of its zig-zag product with G. Note that this entire computation amounts to two recursive calls to computations regarding G_{i-1} (and two computations that correspond to the constant graph G). But since the recursion depth is logarithmic in the size of the final graph (i.e., $t = \log_D |\text{vertices}(G_t)|$), the total number of recursive calls is polynomial in the size of the final graph (and thus the entire computation is polynomial in the size of the final graph). This suffices for the minimal (i.e., "mild") notion of explicitness, but not for the strong one.

The strongly explicit version. To achieve a *strongly explicit construction*, we slightly modify the iterative construction. Rather than letting $G_{i+1} = G_i^2 ⓩ G$, we let $G_{i+1} = (G_i \times G_i)^2 ⓩ G$, where $G' \times G'$ denotes the *tensor product of G' with itself*; that is, if $G' = (V', E')$ then $G' \times G' = (V' \times V', E'')$, where

$$E'' = \{\{\langle u_1, u_2\rangle, \langle v_1, v_2\rangle\} : \{u_1, v_1\}, \{u_2, v_2\} \in E'\}$$

(i.e., $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ are connected in $G' \times G'$ if for $i = 1, 2$ it holds that u_i is connected to v_i in G'). The corresponding edge-rotation function is

$$R''(\langle u_1, u_2\rangle, \langle i_1, i_2\rangle) = (\langle v_1, v_2\rangle, \langle j_1, j_2\rangle),$$

where $R'(u_1, i_1) = (v_1, j_1)$ and $R'(u_2, i_2) = (v_2, j_2)$. We still use $G_1 = G^2$, where (as before) G is d-regular and $\bar{\lambda}(G) < 1/4$, but here G has $D = d^8$ vertices.[10] Using the fact that tensor product preserves the relative eigenvalue-bound while squaring the degree (and the number of vertices), we note that the modified iteration $G_{i+1} = (G_i \times G_i)^2 ⓩ G$ yields a d^2-regular graph with $(D^{2^i-1})^2 \cdot D = D^{2^{i+1}-1}$ vertices, and that $\bar{\lambda}(G_{i+1}) < 1/2$ (because $\bar{\lambda}((G_i \times G_i)^2 ⓩ G) \leq \bar{\lambda}(G_i)^2 + \bar{\lambda}(G)$). Computing the neighbor of a vertex in G_{i+1} boils down to a constant number of such computations regarding G_i, but due to the tensor product operation the depth of the recursion is only double-logarithmic in the size of the final graph (and hence logarithmic in the length of the description of vertices in this graph).

[10] The reason for the change is that $(G_i \times G_i)^2$ will be d^8-regular, since G_i will be d^2-regular.

Digest. In the first construction, the zig-zag product was used both in order to increase the size of the graph and to reduce its degree. However, as indicated by the second construction (where the tensor product of graphs is the main vehicle for increasing the size of the graph), the primary effect of the zig-zag product is reducing the graph's degree, and the increase in the size of the graph is merely a side-effect.[11] In both cases, graph squaring is used in order to compensate for the modest increase in the relative eigenvalue-bound caused by the zig-zag product. In retrospect, the second construction is the "correct" one, because it decouples three different effects, and uses a natural operation to obtain each of them: Increasing the size of the graph is obtained by tensor product of graphs (which in turn increases the degree), the desired degree reduction is obtained by the zig-zag product (which in turn slightly increases the relative eigenvalue-bound), and graph squaring is used in order to reduce the relative eigenvalue-bound.

Advanced topic: Stronger bound regarding the effect of the zig-zag product. In the foregoing description we relied on the fact, proved in [91], that the relative eigenvalue-bound of the zig-zag product is upper-bounded by the sum of the relative eigenvalue-bounds of the two graphs (i.e., $\bar{\lambda}(G'ⓩG) \leq \bar{\lambda}(G') + \bar{\lambda}(G)$)). Actually, a stronger upper-bound is proved in [91]: It holds that $\bar{\lambda}(G'ⓩG) \leq f(\bar{\lambda}(G'), \bar{\lambda}(G)))$, where

$$f(x,y) \stackrel{\text{def}}{=} \frac{(1-y^2)\cdot x}{2} + \sqrt{\left(\frac{(1-y^2)\cdot x}{2}\right)^2 + y^2} \tag{5.7}$$

Indeed, $f(x,y) \leq (1-y^2)\cdot x + y \leq x + y$. On the other hand, for $x \leq 1$, we have $f(x,y) \leq \frac{(1-y^2)\cdot x}{2} + \frac{1+y^2}{2} = 1 - \frac{(1-y^2)\cdot(1-x)}{2}$, which implies

$$\bar{\lambda}(G'ⓩG) \;\leq\; 1 - \frac{(1-\bar{\lambda}(G)^2)\cdot(1-\bar{\lambda}(G'))}{2}\,. \tag{5.8}$$

Thus, $1-\bar{\lambda}(G'ⓩG) \geq (1-\bar{\lambda}(G)^2)\cdot(1-\bar{\lambda}(G'))/2$, and it follows that the zig-zag product has a positive eigenvalue-gap if both graphs have positive eigenvalue-gaps (i.e., $\lambda(G'ⓩG) < 1$ if both $\lambda(G) < 1$ and $\lambda(G') < 1$). Furthermore, if $\bar{\lambda}(G) < 1/\sqrt{3}$ then $1 - \bar{\lambda}(G'ⓩG) > (1 - \bar{\lambda}(G'))/3$. This fact plays an important role in the work of [90] (cf. [44, Sec. 5.2.4]).

[11] We mention that this side-effect may actually be undesired in some applications. For example, in the context of [90] (cf. [44, Sec. 5.2.4]) we would rather not have the graph grow in size, but we can tolerate the constant size blow-up (caused by zig-zag product with a constant-size graph).

Lecture 6

Sampling

We tentatively include two texts: a brief overview (i.e., Section 6.1) and a more detailed overview (i.e., Sections 6.2–6.9).

6.1 A brief overview

In many settings repeated sampling is used to estimate the average of a huge set of values. Namely, a "value" function $\nu:\{0,1\}^n\to\mathbb{R}$ is defined over a huge domain, and one wishes to approximate $\bar{\nu} \stackrel{\rm def}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\nu(x)$ without having to inspect the value of ν at each point in the domain. The obvious thing to do is to sample the domain at random, and obtain an approximation to $\bar{\nu}$ by taking the average of the values of ν on the sample points. It turns out that certain "pseudorandom" sequences of sample points may serve almost as well as truly random sequences of sample points.

Formal Setting. It is essential to have the range of ν be bounded (or else no reasonable approximation is possible). For simplicity, we adopt the convention of having $[0,1]$ be the range of ν, and the problem for other (predetermined) ranges can be treated analogously. Our notion of approximation depends on two parameters: *accuracy* (denoted ε) and *error probability* (denoted δ). We wish to have an algorithm that, with probability at least $1-\delta$, gets within ε of the correct value. This leads to the following definition.

Definition 6.1 (sampler): *A* sampler *is a randomized algorithm that on input parameters* n (length), ε (accuracy) *and* δ (error), *and oracle access to* any *function* $\nu:\{0,1\}^n\to[0,1]$, *outputs, with probability at least* $1-\delta$, *a value that is at most* ε *away from* $\bar{\nu}\stackrel{\rm def}{=}\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\nu(x)$. *Namely,*

$$\mathsf{Pr}[|\mathrm{sampler}^{\nu}(n,\varepsilon,\delta)-\bar{\nu}|>\varepsilon] \;<\; \delta$$

where the probability is taken over the internal coin tosses of the sampler, also called its random seed.

A non-adaptive sampler *is a sampler that consists of two deterministic algorithms: a* sample generating *algorithm, G, and a* evaluation algorithm*, V. On input n, ε, δ and a random* seed*, algorithm G generates a sequence of queries, denoted $s_1, ..., s_m \in \{0,1\}^n$. Algorithm V is given the corresponding ν-values (i.e., $\nu(s_1), ..., \nu(s_m)$) and outputs an estimate to $\bar{\nu}$.*

We are interested in "the complexity of sampling" quantified as a function of the parameters n, ε and δ. Specifically, we will consider three complexity measures: The sample complexity (i.e., the number of oracle queries made by the sampler); the randomness complexity (i.e., the length of the random seed used by the sampler); and the computational complexity (i.e., the running-time of the sampler). We say that a sampler is efficient if its running-time is polynomial in the total length of its queries (i.e., polynomial in both its sample complexity and in n). We will focus on efficient samplers. Furthermore, we will focus on efficient samplers that have optimal (up-to a constant factor) sample complexity, and will wish the randomness complexity to be as low as possible.

We note that all positive results to be reviewed refer to non-adaptive samplers, whereas the lower bound hold also for general samplers. For more details see [41, Sec. 3.6.4].

The naive sampler. The straightforward method (or the naive sampler) consists of *uniformly and independently* selecting sufficiently many sample points (queries), and outputting the average value of the function on these points. Using Chernoff Bound it follows that $O(\frac{\log(1/\delta)}{\varepsilon^2})$ sample points suffice. The naive sampler is optimal (up-to a constant factor) in its sample complexity, but is quite wasteful in randomness.

It is known that $\Omega(\frac{\log(1/\delta)}{\varepsilon^2})$ samples are needed in any sampler, and that that samplers that make $s(n, \varepsilon, \delta)$ queries require randomness at least $n + \log_2(1/\delta) - \log_2 s(n, \varepsilon, \delta) - O(1)$. These lower bounds are tight (as demonstrated by non-explicit and inefficient samplers). These facts guide our quest for improvements, which is aimed at finding more randomness-efficient ways of *efficiently* generating sample sequences that can be used in conjunction with an appropriate evaluation algorithm V. (We stress that V need not necessarily take the average of the values of the sampled points.)

The pairwise-independent sampler. Using a pairwise-independence generator for generating sample points, along with the natural evaluation algorithm (which outputs the average of the values of these points), we obtain a great saving in the randomness complexity: pairwise-independent sampling uses $2n$ random bits rather than the $\Omega((\log(1/\delta))\varepsilon^{-2} \cdot n)$ coins used by the naive sampler. Using Eq. (1.4) it follows that $O(1/\delta\varepsilon^2)$ samples are sufficient to get accuracy ε with error δ. Thus, for constant $\delta > 0$, the Pairwise-Independent Sampler is optimal up-to a constant factor in both its sample and randomness complexities. However, for small δ (i.e., $\delta = o(1)$), this sampler is wasteful in sample complexity.

The Median-of-Averages sampler. A new idea is required for going further, and a relevant tool – random walks on expander graphs – is needed too. Specifically, we combine the Pairwise-Independent Sampler with the Expander Random Walk Generator to obtain a new sampler. The new sampler uses a random walk on an expander with vertex set $\{0,1\}^{2n}$ to generate a sequence of $t \stackrel{\text{def}}{=} O(\log(1/\delta))$ related seeds for t invocations of the Pairwise-Independent Sampler, where each of these invocations uses the corresponding $2n$ bits to

generate a sequence of $O(1/\varepsilon^2)$ samples in $\{0,1\}^n$. Furthermore, each of these invocations returns a value that, with probability at least 0.9, is ε-close to $\bar{\nu}$. The Expander Random Walk Theorem is used to show that, with probability at least $1-\exp(-t) = 1-\delta$, most of these t invocations return an ε-close approximation. Hence, the median among these t values is an (ε,δ)-approximation to the correct value. The resulting sampler, called the Median-of-Averages Sampler, has sample complexity $O(\frac{\log(1/\delta)}{\varepsilon^2})$ and randomness complexity $2n + O(\log(1/\delta))$, which is optimal up-to a constant factor in both complexities.

Further improvements. The randomness complexity of the Median-of-Averages Sampler can be improved from $2n + O(\log(1/\delta))$ to $n + O(\log(1/\delta\varepsilon))$, while maintaining its (optimal) sample complexity (of $O(\frac{\log(1/\delta)}{\varepsilon^2})$). This is done by replacing the Pairwise Independent Sampler by a sampler that picks a random vertex in a suitable expander and samples all its neighbors.

Averaging Samplers. Averaging (a.k.a. Oblivious) samplers are non-adaptive samplers in which the evaluation algorithm is the natural one: that is, it merely outputs the average of the values of the sampled points. Indeed, the Pairwise-Independent Sampler is an averaging sampler, whereas the Median-of-Averages Sampler is not. Interestingly, averaging samplers have applications for which general samplers (and even general non-adaptive samplers) do not suffice. Averaging samplers are closely related to randomness extractors, defined and discussed in Section 8.

An odd perspective. Recall that a non-adaptive sampler consists of a sample generator G and an evaluator V such that for every $\nu:\{0,1\}^n\to[0,1]$ it holds that

$$\mathsf{Pr}_{(s_1,...,s_m)\leftarrow G(U_k)}[|V(\nu(s_1),...,\nu(s_m)) - \bar{\nu}| > \varepsilon] \;<\; \delta.$$

Thus, we may view G as a pseudorandom generator that is subjected to a distinguishability test that is determined by a fixed algorithm V and an arbitrary function $\nu:\{0,1\}^n\to[0,1]$, where we assume that $\mathsf{Pr}[|V(\nu(U_n^{(1)}),...,\nu(U_n^{(m)})) - \bar{\nu}| > \varepsilon] \;<\; \delta$. What is a bit odd here is that, except for the case of averaging samplers, the distinguishability test contains a central component (i.e., the evaluator V) that is potentially custom-made to help the generator G pass the test.[1]

6.2 Introduction to the more detailed overview

We consider the problem of estimating the average of a huge set of values. That is, given oracle access to an arbitrary function $f : \{0,1\}^n \to [0,1]$, we wish to estimate $2^{-n}\sum_{x\in\{0,1\}^n} f(x)$ upto an additive error of ϵ. We are allowed to employ a randomized algorithm that may err with probability at most δ.

We survey known algorithms for this problem and focus on the ideas underlying their construction. In particular, we present an algorithm that makes $O(\epsilon^{-2}\cdot\log(1/\delta))$ queries

[1] Another aspect in which samplers differ from the various pseudorandom generators is in the aim to minimize, rather than maximize, the number of blocks (denoted here by m) in the output sequence. However, also in case of samplers the aim is to maximize the block-length (denoted here by n).

and uses $n + O(\log(1/\epsilon)) + O(\log(1/\delta))$ coin tosses, both complexities being very close to the corresponding lower bounds.

6.2.1 Motivation

In many settings repeated sampling is used to estimate the average value of a huge set of values. Namely, one has access to a value function ν, which is defined over a huge space (say, $\nu\colon\{0,1\}^n \to [0,1]$), and wishes to approximate $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\nu(x)$ without having to inspect the value of ν on the entire domain. It is well-known that sampling ν at sufficiently many (random) points yields such an approximation, but we are interested in the complexity of the approximation. Specifically, (1) how many samples are required? (2) how much randomness is required to generate these samples? and (3) is this generation procedure efficient?

We comment that it is essential to have the range of ν be bounded (or else no reasonable approximation may be possible). Our convention of having $[0,1]$ be the range of ν is adopted for simplicity, and the problem for other (predetermined) ranges can be treated analogously.

6.2.2 Formal Setting

Our notion of approximation depends on two parameters: *accuracy* (denoted ε) and *error probability* (denoted δ). We wish to have an algorithm that, with probability at least $1-\delta$, gets within ε of the correct value. This leads to the following definition.

Definition 6.2 (sampler): *A* sampler *is a randomized algorithm that on input parameters* n (length)*,* ε (accuracy) *and* δ (error)*, and oracle access to* any *function* $\nu : \{0,1\}^n \to [0,1]$*, outputs, with probability at least* $1-\delta$*, a value that is at most* ε *away from* $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\nu(x)$*. Namely,*

$$\mathsf{Pr}\left[|\text{sampler}^{\nu}(n,\varepsilon,\delta) - \bar{\nu}| > \varepsilon\right] \;<\; \delta, \tag{6.1}$$

where the probability is taken over the internal coin tosses of the sampler.

We are interested in "the complexity of sampling" quantified as a function of the parameters n, ε and δ. Specifically, we will consider three complexity measures:

1. Sample Complexity: The number of oracle queries made by the sampler.
2. Randomness Complexity: The number of (unbiased) coin tosses performed by the sampler.
3. Computational Complexity: The running-time of the sampler.

   We say that a sample is efficient if its running-time is polynomial in the total length of its queries (i.e., polynomial in both its sample complexity and in the length parameter, n).

We will focus on efficient samplers. Furthermore, we will focus on efficient samplers that have optimal (upto a constant factor) sample complexity, and will be interested in having the randomness complexity be as low as possible.

6.2.3 Overview and organization of the rest of this chapter

The straightforward method (or the naive sampler) consists of *uniformly and independently* selecting sufficiently many sample points (queries), and outputting the average value of the function on these points. Using Chernoff Bound one can easily show that $O(\frac{\log(1/\delta)}{\varepsilon^2})$ sample points suffice. The naive sampler is optimal (upto a constant factor) in its sample complexity, but is quite wasteful in randomness. In Section 6.3, we discuss the naive sampler and present lower (and upper) bounds on the sample and randomness complexities of samplers. These bounds will guide our quest for improvements.

Pairwise-independent sampling yields a great saving in the randomness complexity. In Section 6.4 we present the Pairwise-Independent Sampler, and discuss its advantages and disadvantages. Specifically, for constant $\delta > 0$, the Pairwise-Independent Sampler is optimal upto a constant factor in both its sample and randomness complexities. However, for small δ (i.e., $\delta = o(1)$), its sample complexity is wasteful.

An additional idea is required for going further, and a relevant tool – random walks on expander graphs (see Lecture 5) – is also used. In Section 6.5, we combine the Pairwise-Independent Sampler with the Expander Random Walk Technique to obtain a new sampler. Loosely speaking, the new sampler uses a random walk on an expander to generate a sequence of $\ell \stackrel{\text{def}}{=} O(\log(1/\delta))$ (related) random pads for ℓ invocations of the Pairwise-Independent Sampler. Each of these invocations returns an ε-close approximation with probability at least 0.99. The expander walk technique yields that, with probability at least $1-\exp(-\ell) = 1-\delta$, most of these ℓ invocations return an ε-close approximation. Thus, the median value is an (ε, δ)-approximation to the correct value (i.e., an approximation that, with probability at least $1-\delta$, is within an additive term of ε of the correct value). The resulting sampler, called the Median-of-Averages Sampler, has sample complexity $O(\frac{\log(1/\delta)}{\varepsilon^2})$ and randomness complexity $2n + O(\log(1/\delta))$.

In Section 6.6 we present an alternative sampler that improves over the pairwise-independent sampler. Maintaining the sample complexity of the latter (i.e., $O(1/\delta\varepsilon^2)$), the new sampler has randomness complexity $n+O(\log(1/\delta\varepsilon))$ (rather than $2n$). Combining this new sampler with the Expander Random Walk Technique, we obtain sample complexity $O(\frac{\log(1/\delta)}{\varepsilon^2})$ and randomness complexity $n + O(\log(1/\delta)) + O(\log(1/\varepsilon))$. Better bounds are obtained for the case of "Boolean samplers" (i.e., algorithms that must only well-approximate Boolean functions). In addition, in Section 6.6 we present two general techniques for improving existing samplers.

We conclude with some open problems (see Section 6.7). In particular, we discuss the notion of "oblivious" (or "averaging") samplers, which is closely related to the notion of randomness extractors (see Section 6.8.2 and more details in [94]).[2] Section 6.8 sketches the outline of an alternative survey that focuses on the notion of "averaging" samplers and on their relation to general samplers, on the one hand, and to randomness extractors, on the other hand.

[2]Indeed, the current text focuses on general samplers, which are not necessarily of the "averaging" type (e.g., the aforementioned Median-of-Averages Sampler). Thus, this survey barely mentions the vast body of work that focuses on randomness extractors, and the interested reader is indeed referred to [94].

The Hitting Problem. In order to distinguish the all-zero function from a function having at least an ε fraction of non-zero values, the sampler must query the function at a non-zero value (or "hit" some non-zero value). Thus, any sampler solves the *hitting problem*, as surveyed in Section 6.9. That is, given an oracle to a Boolean function having at least an ε fraction of 1's, the "hitter" is required to find an input that evaluates to 1. As noted above, each sampler can be used for this purpose, but this is an over-kill. Indeed, all results and techniques regarding samplers (presented in the main text of this survey) have simpler analogues for the hitting problem. Thus, Section 6.9 can be read as a warm-up towards the rest of the survey.

6.3 The Information Theoretic Perspective

The Naive Sampler, presented below, corresponds to the information theoretical (or statistician) perspective of the problem. We augment it by a lower bound on the *sample complexity* of samplers, which is in the spirit of these areas. We conclude with lower and upper bounds on the *randomness complexity* of samplers. The latter lower bound is also information theoretic in nature, but it refers to a concern that is more common in computer science.

6.3.1 The Naive Sampler

The straightforward sampling method consists of randomly selecting a small sample set S and outputting $\frac{1}{|S|}\sum_{x\in S}\nu(x)$ as an estimate to $\bar{\nu}$. More accurately, we select m *independently and uniformly distributed* strings in $\{0,1\}^n$, denoted $s_1,...,s_m$, and output $\frac{1}{m}\sum_{i=1}\nu(s_i)$ as our estimate. Setting $m=\frac{\ln(2/\delta)}{2\varepsilon^2}$, we refer to this procedure as to the Naive Sampler.

To analyze the performance of the Naive Sampler, we use the Chernoff Bound. Specifically, we define m independent random variables, denoted $\zeta_1,...,\zeta_m$, such that $\zeta_i \stackrel{\text{def}}{=} \nu(s_i)$, where the s_i's are independently and uniformly distributed in $\{0,1\}^n$. By Chernoff Bound:

$$\mathsf{Pr}\left[\left|\bar{\nu}-\frac{1}{m}\sum_{i=1}^{m}\zeta_i\right|>\varepsilon\right] \leq 2\exp\left(-2\varepsilon^2 m\right) \quad (6.2)$$

$$= \delta \quad (6.3)$$

where Eq. (6.3) is due to $m=\ln(2/\delta)/2\varepsilon^2$. Observing that $\frac{1}{m}\sum_{i=1}^{m}\zeta_i$ represents the estimate output by the Naive Sampler, we have established that the Naive Sampler indeed satisfies Definition 6.2 (i.e., is indeed a sampler). We now consider the complexity of the Naive Sampler

- Sample Complexity: $m \stackrel{\text{def}}{=} \frac{\ln(2/\delta)}{2\varepsilon^2} = \Theta(\frac{\log(1/\delta)}{\varepsilon^2})$.
- Randomness Complexity: $m\cdot n = \Theta(\frac{\log(1/\delta)}{\varepsilon^2}\cdot n)$.
- Computational Complexity: indeed efficient.

In light of Theorem 6.3 (below), the sample complexity of the Naive Sampler is optimal upto a constant factor. However, as we will shortly see, it is extremely wasteful in its usage

of randomness. In fact, the rest of this survey is devoted to presenting ways for redeeming the latter aspect.

6.3.2 A Sample Complexity Lower Bound

We first assert that the Naive Sampler is quite good as far as sample complexity is concerned. The following theorem is analogous to many results known in statistics, though we are not aware of a reference prior to [27] where it can be found.

Theorem 6.3 [27]: *Any sampler has sample complexity bounded below by*

$$\min\left\{2^{(n-4)/2}, \frac{\ln(1/O(\delta))}{4\varepsilon^2}\right\}$$

provided $\varepsilon \leq \frac{1}{8}$ *and* $\delta \leq \frac{1}{6}$.

Note that a (constant factor) gap remains between the lower bound asserted here and the upper bound established by the Naive Sampler. We conjecture that the lower bound can be improved. Motivated by the lower bound, we say that a sampler is sample-optimal if its sample complexity is $O(\frac{\log(1/\delta)}{\varepsilon^2})$.

6.3.3 Randomness Complexity Lower and Upper Bounds

We first assert that the Naive Sampler is quite bad as far as randomness complexity is concerned. First evidence towards our claim is provided by a non-explicit (and so inefficient) sampler:

Theorem 6.4 [27]: *There exists a* (non-efficient) *sampler with sample complexity* $\frac{2\ln(4/\delta)}{\varepsilon^2}$ *and randomness complexity* $n + 2\log_2(2/\delta) + \log_2\log_2(1/\varepsilon)$.

The proof is by a probabilistic argument that, given the Naive Sampler, asserts the existence of a relatively small set of possible coin tosses under which this sampler behaves almost as under all possible coin tosses (with respect to any possible function ν). Actually, the randomness bound can be improved to $n + \log_2(1/\delta) - \log_2\log_2(1/\delta)$ while using a constant factor larger sample complexity and more sophisticated techniques [117]. More generally:

Theorem 6.5 [117]: *For every function* $s : [0,1]^2 \to \mathbb{R}$ *such that* $s(\varepsilon,\delta) \geq \frac{2\log_2(1/\delta)}{\varepsilon^2}$, *there exists a* (non-efficient) *sampler with sample complexity* $s(\varepsilon,\delta)$ *and randomness complexity*

$$n + \log_2(1/\delta) + 2\log_2(4/\varepsilon) - \log_2 s(\varepsilon,\delta)$$

This gets us very close to the following lower bound.

Theorem 6.6 [27]: *Let* $s : \mathbb{N} \times [0,1]^2 \to \mathbb{R}$. *Any sampler that has sample complexity at most* $s(n,\varepsilon,\delta)$, *has randomness complexity at least*

$$n + \log_2(1/\delta) - \log_2 s(n,\varepsilon,\delta) - \log_2(1-2\varepsilon)^{-1} - 2,$$

provided $\varepsilon, \delta < 0.5$ *and* $s(n,\varepsilon,\delta) \leq 2^{n-1}$.

The dependency of the lower bound on the sample complexity should not come as a surprise. After all, there exists a deterministic sampler that queries the function on the entire domain. Furthermore, the upper bound of Theorem 6.5 does express a similar trade-off between randomness complexity and sample complexity. Similarly, one should not be surprised at the effect of $1-2\varepsilon$ on the bound: For example, when $\varepsilon = 0.5$, a sample may merely output $\widetilde{\nu} = \frac{1}{2}$ as its estimate and always be within ε of the average of any function $\nu : \{0,1\}^n \to [0,1]$.

Using Theorem 6.6, we obtain a lower bound on the randomness complexity of any *sample-optimal* sampler:

Corollary 6.7 [27]: *Any sampler that has sample complexity* $O(\frac{\log(1/\delta)}{\varepsilon^2})$*, has randomness complexity at least*[3]

$$n + (1 - o(1)) \cdot \log_2(1/\delta) - 2\log_2(1/\varepsilon),$$

provided $\varepsilon, \delta < 0.4$ *and* $\frac{\log(1/\delta)}{\varepsilon^2} = o(2^n)$.

6.4 The Pairwise-Independent Sampler

To motivate the Pairwise-Independent Sampler, let us confront two well-known central limit theorems: Chernoff Bound, which refers to *totally independent* random variables, and Chebyshev's Inequality, which refers to *pairwise-independent* random variables

Chernoff Bound: Let $\zeta_1, ..., \zeta_m$ be *totally independent* random variables, each ranging in $[0,1]$ and having expected value μ. Then,

$$\mathsf{Pr}\left[\left|\mu - \frac{1}{m}\sum_{i=1}^{m}\zeta_i\right| > \varepsilon\right] \;\leq\; 2\exp\left(-2\varepsilon^2 m\right)$$

Chebyshev's Inequality: Let $\zeta_1, ..., \zeta_m$ be *pairwise-independent* random variables, each ranging in $[0,1]$ and having expected value μ. Then,

$$\mathsf{Pr}\left[\left|\mu - \frac{1}{m}\sum_{i=1}^{m}\zeta_i\right| > \varepsilon\right] \;\leq\; \frac{1}{4\varepsilon^2 m}$$

Our conclusion is that these two bounds essentially agree when $m = O(1/\varepsilon^2)$. That is, in both cases $\Theta(1/\varepsilon^2)$ identical random variables are necessary and sufficient to guarantee a concentration within ε with constant probability. Thus, if this is what we want, then there is no point in using the more sophisticated Chernoff Bound, which requires more of the random variables.

In the context of sampling, our conclusion is that for achieving an approximation to within ε accuracy with constant error probability, using $O(1/\varepsilon^2)$ pairwise-independent random sample points is as good as using $O(1/\varepsilon^2)$ totally independent random sample points. Furthermore, in the first case we may be save a lot in terms of randomness.

[3] The $o(1)$ term is actually $\frac{\log_2 O(\log(1/\delta))}{\log_2(1/\delta)}$.

The Pairwise-Independent Sampler [30]: On input parameters n, ε and δ, set $m \stackrel{\text{def}}{=} \frac{1}{4\varepsilon^2\delta}$ and generate a sequence of m *pairwise-independently and uniformly distributed* strings in $\{0,1\}^n$, denoted $s_1, ..., s_m$. Using the oracle access to ν, output $\frac{1}{m}\sum_{i=1}\nu(s_i)$ as the estimate to $\bar{\nu}$. Using Chebyshev's Inequality, one can easily see that the Pairwise-Independent Sampler indeed satisfies Definition 6.2 (i.e., is indeed a sampler).

There are two differences between the Naive Sampler and the Pairwise-Independent Sampler. Whereas the former uses independently selected sample points, the latter uses a sequence of pairwise independent sample points. As we shall see, this allows the latter sampler to use much less randomness. On the other hand, the Naive Sampler uses $O(\frac{\log(1/\delta)}{\varepsilon^2})$ samples (which is optimal upto a constant factor), whereas the Pairwise-Independent Sampler uses $O(\frac{1}{\varepsilon^2\delta})$ samples. However, for constant δ, both samplers use essentially the same number of sample points. Thus, for constant δ, the Pairwise-Independent Sampler offers a saving in randomness while being sample-optimal.

Generating a Pairwise-Independent sequence: Whereas generating m totally independent random points in $\{0,1\}^n$ requires $m \cdot n$ unbiased coin flips, one can generate m ($m \leq 2^n$) pairwise-independent random points using only $O(n)$ unbiased coin flips. We present two well-known ways of doing this.

1. Linear functions over finite fields: We associate $\{0,1\}^n$ with the finite field $F \stackrel{\text{def}}{=} \mathrm{GF}(2^n)$. Let $\alpha_1, ..., \alpha_m$ be $m \leq |F|$ distinct elements of F. To generate a (pairwise-independent) sequence of length m, we uniformly and independently select $s, r \in F$, and let the i^{th} element in the sequence be $e_i \stackrel{\text{def}}{=} r + \alpha_i s$ (where the arithmetic is that of F). The fact that this construction yields a pairwise-independent sequence is proved in Theorem 3.1. Only $2n$ random coins are required in this construction, but the drawback is that we need a representation of the field F (i.e., an irreducible polynomial of degree n over GF(2)) which may not be easy to find in general.[4] Still, for specific values of n a good representation exists: Specifically, for $n = 2 \cdot 3^\ell$ (with ℓ integer), the polynomial $x^n + x^{n/2} + 1$ is irreducible [51, p. 96], and so we obtain a representation of $\mathrm{GF}(2^n)$ for such n's.

2. Toeplitz matrices: To avoid problems with non-trivial representation, one may use the following construction. We associate $\{0,1\}^n$ with the n-dimensional vector space over GF(2). Let $v_1, ..., v_m$ be $m \leq 2^n$ distinct vectors in this vector space. A Toeplitz matrix is a matrix with all diagonals being homogeneous; that is, $T = (t_{i,j})$ is a Toeplitz matrix if $t_{i,j} = t_{i+1,j+1}$, for all i, j. Note that a Toeplitz matrix is determined by its first row and first column (i.e., the values of $t_{1,j}$'s and $t_{i,1}$'s). To generate a (pairwise-independent) sequence of length m, we uniformly and independently select an n-by-n Boolean Toeplitz matrix, T, and an n-dimensional Boolean vector u. We let the i^{th} element in the sequence be $e_i \stackrel{\text{def}}{=} Tv_i + u$ (where the arithmetic is that of the vector space). The fact that this construction yields a pairwise-independent sequence is proved in Theorem 3.2. Here, we merely note that $3n - 1$ random coins suffice for this construction,

[4] Things are not better if we wish to work with a large field of prime cardinality; since we need to find such a prime.

Plugging-in either of these constructions, we obtain the following complexities for the Pairwise-Independent Sampler

- Sample Complexity: $\frac{1}{4\delta\varepsilon^2}$.
- Randomness Complexity: $2n$ or $3n-1$, depending on which of the constructions is used.
- Computational Complexity: Indeed efficient.

We note that for constant δ, the sample and randomness complexities match the lower bounds upto a constant factor. However, as δ decreases, the sample complexity of the Pairwise-Independent Sampler increases faster than the corresponding complexity of the Naive Sampler. Redeeming this state of affairs is our next goal.

6.5 The (Combined) Median-of-Averages Sampler

Our goal here is to decrease the sample complexity of the Pairwise-Independent Sampler while essentially maintaining its randomness complexity. To motivate the new construction we first consider an oversimplified version of it.

Median-of-Averages Sampler (oversimplified): On input parameters n, ε and δ, set $m \stackrel{\text{def}}{=} \Theta(\frac{1}{\varepsilon^2})$ and $\ell \stackrel{\text{def}}{=} \Theta(\log(1/\delta))$, generate ℓ independent m-element sequences, each being a sequence of m *pairwise-independently and uniformly distributed* strings in $\{0,1\}^n$. Denote the sample points in the i^{th} sequence by $s^i_1, ..., s^i_m$. Using the oracle access to ν, compute $\widetilde{\nu}^i \stackrel{\text{def}}{=} \frac{1}{m}\sum_{j=1}^{m}\nu(s^i_j)$, for $i = 1, ..., \ell$, and output the *median value* among these $\widetilde{\nu}^i$'s. Using Chebyshev's Inequality (as in previous section), for each i, it holds that

$$\mathsf{Pr}[|\widetilde{\nu}^i - \bar{\nu}| > \varepsilon] < 0.1$$

and so

$$\begin{aligned}\mathsf{Pr}\left[|\{i : |\widetilde{\nu}^i - \bar{\nu}| > \varepsilon\}| \geq \frac{\ell}{2}\right] &< \sum_{j=\ell/2}^{\ell}\binom{\ell}{j}\cdot 0.1^j \cdot 0.9^{\ell-j} \\ &< 2^\ell \cdot 0.1^{\ell/2} \\ &\leq \delta,\end{aligned}$$

where the last inequality is due to the choice of ℓ. Thus, the oversimplified version described above is indeed a sampler and has the following complexities

- Sample Complexity: $\ell \cdot m = O(\frac{\log(1/\delta)}{\varepsilon^2})$.
- Randomness Complexity: $\ell \cdot O(n) = O(n \cdot \log(1/\delta))$.
- Computational Complexity: Indeed efficient.

Thus, the sample complexity is optimal (upto a constant factor), but the randomness complexity is higher than what we aim for. To reduce the randomness complexity, we use the same approach as above, but take dependent sequences rather than independent ones. The dependency we use is such that essentially preserves the probabilistic behavior of independent choices. Specifically, we use random walks on expander graphs (cf. Lecture 5). to generate a sequence of ℓ "seeds" each of length $O(n)$. Each seed is used to generate a sequence of m pairwise independent elements in $\{0,1\}^n$, as above. Let us generalize this construction as follows.

Theorem 6.8 (general median-composition [16]): *Suppose we are given an efficient sampler of sample complexity $s(n,\varepsilon,\delta)$ and randomness complexity $r(n,\varepsilon,\delta)$. Then:*

1. *There exists an efficient sampler with sample complexity $O(s(n,\varepsilon,0.01)\cdot\log(1/\delta))$ and randomness complexity $r(n,\varepsilon,0.01)+O(\log(1/\delta))$.*
2. *For any $c>4$, there exists an $\alpha>0$ and an efficient sampler with sample complexity $O(s(n,\varepsilon,\alpha)\cdot\log(1/\delta))$ and randomness complexity $r(n,\varepsilon,\alpha)+c\cdot\log_2(1/\delta)$.*

Proof: For Item 1, let $r \stackrel{\text{def}}{=} r(n,\varepsilon,0.01)$. We use an explicit construction of expander graphs with vertex set $\{0,1\}^r$, degree d and second eigenvalue λ so that $\lambda/d<0.1$. We consider a random walk of (edge) length $\ell-1=O(\log(1/\delta))$ on this expander, and use each of the ℓ vertices along the path as random coins for the given sampler. Thus, we obtain ℓ estimates to $\bar{\nu}$ and output the median value as the estimate of the new sampler. To analyze the performance of the resulting sampler, we let W denote the set of coin tosses (for the basic sampler) that make the basic sampler output an estimate that is ε-far from the correct value (i.e., $\bar{\nu}$). Thus, W denotes the set of coin tosses that are bad for the basic sampler, and by the hypothesis $\frac{|W|}{2^r}\leq 0.01$. Using (Part 2 of) Lemma 5.5 (with some W_i's set to W and the others set to $\{0,1\}^r$), we infer that the probability that at least $\ell/2$ vertices of the path reside in W is smaller than

$$\begin{aligned}\sum_{j=\ell/2}^{\ell}\binom{\ell}{j}\cdot 0.02^{j/2} &< 2^{\ell}\cdot 0.02^{\ell/4}\\ &\leq \delta.\end{aligned}$$

Note that we have used $\ell\cdot s(n,\varepsilon,0.01)$ samples and $r+(\ell-1)\cdot\log_2 d = r+O(\log(1/\delta))$ coin tosses. Item 1 follows.

Item 2 is proved using the same argument but using Ramanujan Graphs (and slightly more care). Specifically, we use Ramanujan graphs (i.e., expanders with $\lambda\leq 2\sqrt{d-1}$) with vertex set $\{0,1\}^r$, where $r\stackrel{\text{def}}{=} r(n,\varepsilon,\alpha)$ and $\alpha=(\frac{\lambda}{d})^2$. Repeating the foregoing argument, with $\ell-1=\frac{2\log_2(1/\delta)}{\log_2(\alpha/8)}$, we obtain an efficient sampler that uses $\ell\cdot s(n,\varepsilon,\alpha)$ samples and $r+(\ell-1)\cdot\log_2 d = r+(4+\frac{16}{(\log_2 d)-8})\cdot\log_2(1/\delta)$ coin tosses. Since this can be done with a sufficiently large d, Item 2 follows. ■

Combining the Pairwise-Independent Sampler with Theorem 6.8, we get

Corollary 6.9 (The Median-of-Averages Sampler [16]): *There exists an efficient sampler with*

- Sample Complexity: $O(\frac{\log(1/\delta)}{\varepsilon^2})$.
- Randomness Complexity: $O(n + \log(1/\delta))$.

Furthermore, we can obtain randomness complexity $2n + (4 + o(1)) \cdot \log_2(1/\delta))$.

In the next section, we further reduce the randomness complexity of samplers (from $2n + O(\log(1/\delta))$ to $n + O(\log(1/\varepsilon) + \log(1/\delta))$, while maintaining the sample complexity (up-to a multiplicative constant).

Generalizing Theorem 6.8. A close look at the proof of Theorem 6.8 reveals the fact that the median value obtained via an expander random walk (on the vertex set $\{0,1\}^r$) is used as a sampler of accuracy 0.49 and error probability δ. This suggests the following generalization of Theorem 6.8: *Suppose we are given two efficient samplers such that the* i^{th} *sampler has sample complexity* $s_i(n, \varepsilon, \delta)$ *and randomness complexity* $r_i(n, \varepsilon, \delta)$. *Then, for every* $\delta_0 \in (0, 0.5)$, *there exists an efficient sampler of sample complexity* $s_2(r, 0.5 - \delta_0, \delta) \cdot s_1(n, \varepsilon, \delta_0)$ *and randomness complexity* $r_2(r, 0.5 - \delta_0, \delta)$, *where* $r \stackrel{\text{def}}{=} r_1(n, \varepsilon, \delta_0)$. Theorem 6.8 is derived as a special case, when using the expander random walk as the second sampler and setting $\delta_0 = 0.01$.

6.6 Advanced Topic: The Expander Sampler and Two Generic Transformations

The main result of this section is the following:

Theorem 6.10 [16, 50]: *There exists an efficient sampler that has*

- Sample Complexity: $O(\frac{\log(1/\delta)}{\varepsilon^2})$.
- Randomness Complexity: $n + \log_2(1/\varepsilon) + O(\log(1/\delta))$.

The theorem is proved by applying Theorem 6.8 to a new efficient sampler that makes $O(\frac{1}{\delta\varepsilon^2})$ oracle queries and tosses $n + \log_2(1/\varepsilon)$ coins. We start by presenting a sampler for the special case of Boolean functions.

Definition 6.11 (Boolean sampler): *A* Boolean sampler *is a randomized algorithm that on input parameters* n, ε *and* δ, *and oracle access to* any Boolean *function* $\nu : \{0,1\}^n \to \{0,1\}$, *outputs, with probability at least* $1 - \delta$, *a value that is at most* ε *away from* $\bar{\nu} \stackrel{\text{def}}{=} \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \nu(x)$. *Namely,*

$$\mathsf{Pr}[|\text{sampler}^{\nu}(n, \varepsilon, \delta) - \bar{\nu}| > \varepsilon] \;<\; \delta$$

where the probability is taken over the internal coin tosses of the sampler.

That is, unlike (general) samplers, a *Boolean sampler* is required to work well only when given access to a *Boolean* function. The rest of this section is organized as follows:

In Section 6.6.1 we present the Expander Sampler, which is a Boolean sampler of sample complexity $O(1/\delta\varepsilon^2)$ and randomness complexity n. This sample complexity is obtained by using Ramanujan Graphs (rather than arbitrary expanders).

In Section 6.6.2 we present a (general) transformation of Boolean samplers to general ones.

In Section 6.6.3 we revisit the Expander Sampler, while using an arbitrary expander. More importantly, we present another generic composition of samplers, and obtain an alternative construction by using this composition in conjunction with the aforementioned sampler. Unlike the composition method that underlies Theorem 6.8, which reduces the error complexity (in an efficient manner), the current composition reduces the sample complexity.

Theorem 6.10 is proved by combining the ideas of Sections 6.6.1 and 6.6.2. An alternative proof of a somewhat weaker result is obtained by combining the ideas of Sections 6.6.1 and 6.6.3.

6.6.1 A Sampler for the Boolean Case

We start by presenting a sampler for the special case of Boolean functions. Our sampling procedure is exactly the one suggested by Karp, Pippinger and Sipser for hitting a witness set [68] (cf. Section 6.9), yet the analysis is somewhat more involved. Furthermore, to get an algorithm that samples the universe only on $O(1/\delta\varepsilon^2)$ points, it is crucial to use a Ramanujan graph in role of the expander in the Karp-Pippinger-Sipser method.

The sampler. We use an expander of degree $d = 4/\delta\epsilon^2$ second eigenvalue bounded by λ and associate the vertex set of the expander with $\{0,1\}^n$. The sampler consists of uniformly selecting a vertex, v, (of the expander) and averaging over the values assigned (by ν) to all the neighbors of v; that is, *the algorithm outputs the estimate*

$$\widetilde{\nu} \stackrel{\text{def}}{=} \frac{1}{d} \sum_{u \in \Gamma(v)} \nu(u), \tag{6.4}$$

where $\Gamma(v)$ *denotes the set of neighbors of vertex* v.

This algorithm has

- Sample Complexity: $O(\frac{1}{\delta\varepsilon^2})$.
- Randomness Complexity: n.
- Computational Complexity: Indeed efficient; that is, polynomial in n, ε^{-1} and δ^{-1}.

Lemma 6.12 [50]: *The foregoing algorithm constitutes an efficient Boolean sampler.*

Proof: We denote by B the set of *bad* choices for the algorithm; namely, the set of vertices that once selected by the algorithm yield a wrong estimate. That is, $v \in B$ if

$$\left| \frac{1}{d} \sum_{u \in \Gamma(v)} \nu(u) - \bar{\nu} \right| > \varepsilon. \tag{6.5}$$

Denote by B' the subset of $v \in B$ for which

$$\frac{1}{d} \sum_{u \in \mathrm{N}(v)} \nu(u) > \bar{\nu} + \varepsilon. \tag{6.6}$$

It follows that each $v \in B'$ has ϵd too many neighbors in the set $A \stackrel{\mathrm{def}}{=} \{u : \nu(u)=1\}$; namely,

$$|\{u \in \mathrm{N}(v) : u \in A\}| > (\rho(A) + \epsilon) \cdot d, \tag{6.7}$$

where $\rho(A) \stackrel{\mathrm{def}}{=} \frac{|A|}{N}$ and $N \stackrel{\mathrm{def}}{=} 2^n$. Using the Expander Mixing Lemma (i.e., Lemma 5.4), we get that

$$\begin{aligned} \varepsilon \cdot \rho(B') &= \left| \frac{|B'| \cdot (\rho(A) + \varepsilon)d}{dN} - \rho(B') \cdot \rho(A) \right| \\ &\leq \left| \frac{|(B' \times A) \cap E|}{|E|} - \frac{|A|}{|V|} \cdot \frac{|B'|}{|V|} \right| \\ &\leq \frac{\lambda}{d} \cdot \sqrt{\rho(A) \cdot \rho(B')}. \end{aligned}$$

Thus,

$$\rho(B') \leq \left(\frac{\lambda}{d\varepsilon} \right)^2 \cdot \rho(A). \tag{6.8}$$

Using $\lambda \leq 2\sqrt{d}$ and $d = \frac{4}{\delta\varepsilon^2}$, we get $\rho(B') \leq \delta \cdot \rho(A)$. Using a similar argument,[5] we can show that $\rho(B \setminus B') \leq \delta \cdot (1 - \rho(A))$. Thus, $\rho(B) \leq \delta$, and the claim follows. ∎

Claim 6.13 [50]: *Observe that if we were to use an arbitrary d-regular graph with second eigenvalue λ, then the foregoing proof would hold provided that*

$$\frac{\lambda}{d} \leq \sqrt{\delta\varepsilon^2}. \tag{6.9}$$

This yields, for any such d-regular graph, an efficient Boolean sampler with sample complexity *d and* randomness complexity *n.*

6.6.2 From Boolean Samplers to General Samplers

The following generic transformation was suggested to us by Luca Trevisan.

Theorem 6.14 (Boolean samplers imply general ones): *Suppose we are given an efficient Boolean sampler of sample complexity $s(n, \varepsilon, \delta)$ and randomness complexity $r(n, \varepsilon, \delta)$. Then, there exists an efficient sampler with sample complexity $s(n + \log_2(1/\varepsilon), \varepsilon/2, \delta)$ and randomness complexity $r(n + \log_2(1/\varepsilon), \varepsilon/2, \delta)$.*

[5] That is, we consider the set $B'' \stackrel{\mathrm{def}}{=} B \setminus B'$, and observe that every $v \in B''$ has εd too many neighbours in $A'' \stackrel{\mathrm{def}}{=} \{0,1\}^n \setminus A$. Hence, we conclude that $\rho(B'') \leq \delta \cdot \rho(A'')$.

Proof: As a mental experiment, given an arbitrary function $\nu : \{0,1\}^n \to [0,1]$, we define a Boolean function $\mu : \{0,1\}^{n+\ell} \to \{0,1\}$, where $\ell \stackrel{\text{def}}{=} \log_2(1/\varepsilon)$, as follows: For every $x \in \{0,1\}^n$ and $i = 1, ..., \varepsilon^{-1}$, we set $\mu(x,i) \stackrel{\text{def}}{=} 1$ if and only if $\nu(x) > (i-0.5)\cdot\varepsilon$ (i.e., iff $i < \varepsilon^{-1}\nu(x) + 0.5$). Then, for every x, it holds that $|\nu(x) - \varepsilon \cdot \sum_{i=1}^{1/\varepsilon} \mu(x,i)| \leq \varepsilon/2$. Thus, if we were to sample μ and obtain an $\varepsilon/2$-approximation of $\bar{\mu}$ then we get an ε-approximation of $\bar{\nu}$. Now, although we don't have actual access to μ we can emulate its answers given an oracle to ν.

Given a Boolean sampler, B, we construct a general sampler, A, as follows. On input n, ε, δ and access to an arbitrary ν as above, algorithm A sets $n' = n + \ell$, $\varepsilon' = \varepsilon/2$, and $\delta' = \delta$, and invoke B on input $n', \varepsilon', \delta'$. When B makes a query $(x,i) \in \{0,1\}^n \times \{0,1\}^\ell$, algorithm A queries for $\nu(x)$ and returns 1 if and only if $\nu(x) > (i-0.5)\cdot\varepsilon$. When B halts with output v, algorithm A does the same. The theorem follows. ■

Combining the sampler of Section 6.6.1 with Theorem 6.14, we get

Corollary 6.15 (The Expander Sampler, revisited): *There exists an efficient sampler that has*

- Sample Complexity: $O(\frac{1}{\delta\varepsilon^2})$.
- Randomness Complexity: $n + \log_2(1/\varepsilon)$.

Theorem 6.10 follows by combining Corollary 6.15 with Theorem 6.8.

6.6.3 An Alternative Construction

Using an arbitrary expander graph (with $d = \text{poly}(1/\varepsilon\delta)$ and $\frac{\lambda}{d} < \sqrt{\delta\varepsilon^2}$) and invoking Comment 6.13, we have an efficient Boolean sampler with sample complexity $\text{poly}(1/\varepsilon\delta)$ and randomness complexity n. Using Theorem 6.14, we get

Corollary 6.16 (The Expander Sampler, revisited again): *There exists an efficient sampler with* sample complexity $\text{poly}(1/\varepsilon\delta)$ *and* randomness complexity $n + \log_2(1/\varepsilon)$.

To derive (a weaker form of) Theorem 6.10 via the foregoing sampler, we first need to reduce its sample complexity. This is done via the following general transformation. We say that a sampler is of the averaging type if its output is the average value obtained on its queries, which in turn are determined as a function of its own coin tosses (independently of the answers obtained on previous queries).

Theorem 6.17 (reducing sample complexity (or "sampling the sample")): *Suppose we are given two efficient samplers such that the i^{th} sampler has sample complexity $s_i(n,\varepsilon,\delta)$ and randomness complexity $r_i(n,\varepsilon,\delta)$. Further suppose that the first sampler is of the averaging type. Then, there exists an efficient sampler of sample complexity $s_2(\log_2 s_1(n,\varepsilon/2,\delta/2),\varepsilon/2,\delta/2)$ and randomness complexity $r_1(n,\varepsilon/2,\delta/2) + r_2(\log_2 s_1(n,\varepsilon/2,\delta/2),\varepsilon/2,\delta/2)$. Furthermore, if also the second sampler is of the averaging type, then so is the resulting sampler.*

Proof: We compose the two samplers as follows. Setting $m \stackrel{\text{def}}{=} s_1(n, \varepsilon/2, \delta/2)$, we invoke the first sampler and determine the m queries it would have asked (given a particular choice of its coins).[6] We then use the second sampler to sample these m queries (invoking it with parameters $\log_2 m, \varepsilon/2$ and $\delta/2$). Specifically, we let the second sampler make virtual queries into the domain $[m] \stackrel{\text{def}}{=} \{1, ..., m\}$ and answer a query $q \in [m]$ by the value of the function at the i^{th} query specified by the first sampler. That is, given access to a function $\nu : \{0,1\}^n \to [0,1]$, and determining a sequence r of coins for the first sampler, we consider the function $\nu_r : [m] \to [0,1]$ defined by letting $\nu_r(i) = \nu(q_{r,i})$ where $q_{r,i}$ is the i^{th} query made by the first sampler on coins r. We run the second sampler providing it virtual access to the function ν_r in the obvious manner, and output its output. Thus, the complexities are as claimed and the combined sampler errs if either $|\bar{\nu} - \frac{1}{m}\sum_{i=1}^m \nu(q_{r,i})| > \frac{\varepsilon}{2}$ or $|\frac{1}{m}\sum_{i=1}^m \nu(q_{r,i}) - \widetilde{\nu}_r| > \varepsilon/2$, where $\widetilde{\nu}_r$ is the estimate output by the second sampler when given virtual access to ν_r. Observing that the first event means that the first sampler errs (here we use the hypothesis that this sampler is averaging) and that the second event means that the second sampler errs (here we use $\sum_{i=1}^m \nu(q_{r,i}) = \bar{\nu}_r$), we are done. ■

It is tempting to try to improve the sample complexity of the sampler asserted in Corollary 6.16 by combining it with the Pairwise-Independent Sampler, via Theorem 6.17. The problem is that the former sampler, which we wish to use in the role of the outer sampler, is not of the averaging type. Indeed, the expander sampler (of Comment 6.13) is of the averaging type, but the proof of Theorem 6.14 does not preserve this feature. Instead, as shown in Theorem 6.19 (below), any Boolean sampler of the averaging type is a general sampler of the averaging time, except that its accuracy and error probability may increase by a constant factor. Thus, combining the sampler of Comment 6.13 with the Pairwise-Independent Sampler, via Theorem 6.17, we obtain:

Corollary 6.18 (sampling the Expander Sampler): *There exists an efficient sampler that has*

- Sample Complexity: $O(\frac{1}{\delta\varepsilon^2})$.
- Randomness Complexity: $n + O(\log(1/\varepsilon)) + O(\log(1/\delta))$.

Indeed, the sampler asserted in Corollary 6.18 operates by selecting a random vertex in an expander and taking a pairwise-independent sample of its neighbor set. A weaker form of Theorem 6.10 (i.e., with an $O(\log(1/\varepsilon)$ term rather than with a $\log_2(1/\varepsilon)$ term) follows by combining Corollary 6.18 with Theorem 6.8.

It is left to establish the aforementioned claim by which any Boolean sampler of the averaging type is a general sampler (of the averaging time), except that its accuracy and error probability may increase by a constant factor. (A similar statement was proved in [117].)

Theorem 6.19 (Boolean vs general samplers of the averaging type): *Every Boolean sampler of the averaging type, having sample complexity $s(n, \varepsilon, \delta)$ and randomness complexity $r(n, \varepsilon, \delta)$, is a general sampler* (of the averaging type) *with sample complexity $s(n, \varepsilon/4, \delta/3)$ and randomness complexity $r(n, \varepsilon/4, \delta/3)$.*

[6] Here we use the hypothesis that the first sampler is non-adaptive; that is, its queries are determined (only) by its coin tosses (independently of the answers obtained on previous queries).

	sample complexity	randomness complexity	pointer
lower bound	$\Omega(\frac{\log(1/\delta)}{\varepsilon^2})$		Thm. 6.3
lower bound	for $O(\frac{\log(1/\delta)}{\varepsilon^2})$	$n + (1 - o(1)) \cdot \log_2(1/\delta) - 2\log_2(1/\varepsilon)$	Cor. 6.7
upper bound	$O(\frac{\log(1/\delta)}{\varepsilon^2})$	$n + \log_2(1/\delta)$	Thm. 6.5
algorithm	$O(\frac{\log(1/\delta)}{\varepsilon^2})$	$n + O(\log(1/\delta)) + \log_2(1/\varepsilon)$	Thm. 6.10
algorithm	$\mathrm{poly}(\varepsilon^{-1}, \log(1/\delta))$	$n + (1 + \alpha) \cdot \log_2(1/\delta)$, $\forall \alpha > 0$	Thm. 6.20

Figure 6.1: Summary of main results.

Proof: For any function $\nu: \{0,1\}^n \to [0,1]$, we consider a random function $\rho: \{0,1\}^n \to \{0,1\}$ such that, for every x, we set $\rho(x) = 1$ with probability $\nu(x)$, independently of the setting of all other arguments. Clearly, with probability $1 - \exp(-2\varepsilon^2 2^n) > 1 - \delta$, it holds that $|\bar{\nu} - \bar{\rho}| < \varepsilon$. Furthermore, fixing any possible outcome of the sampler's coins, with probability at least $1 - \exp(-8\varepsilon^2 s)$ over the choice of ρ, the average of the ρ-values queried by the sampler is 2ε-close to the average of the ν-values, where s denotes the number of queries. Since (by Theorem 6.3) $s > \varepsilon^{-2}\log(1/\delta)/8$, with probability at least $1 - \delta$ over the choice of ρ, the average that the Boolean sampler outputs when given access to ν is 2ε-close to the average it would have output on a random ρ, which in turn (with probability at least $1 - \delta$ over the sampler's coins) is ε-close to $\bar{\rho}$. Thus, with probability at least $1 - 3\delta$ (over the sampler's coins), the Boolean sampler outputs a value that is 4ε-close to $\bar{\nu}$, ■

6.7 Conclusions and Open Problems

The main results surveyed in the text are summarized in Figure 6.1. The first row tabulates $\Omega(\varepsilon^{-2}\log(1/\delta))$ as a lower bound on sample complexity and the subsequent three rows refer to sample-optimal samplers (i.e., samplers of sample complexity $O(\varepsilon^{-2}\log(1/\delta))$). The last row refers to a sampler (cf., Thm. 6.20 below) that has randomness complexity closer to the lower bound. However, this sampler is not sample-optimal.

The randomness complexity of sample-optimal samplers. A closer look at the randomness complexity of sample-optimal samplers is provided in Figure 6.2. The first two rows tabulate lower and upper bounds, which are $2\log_2(1/\varepsilon) + O(1)$ apart. Our conjecture is that the lower bound can be improved to match the upper bound.[7] The efficient samplers use somewhat more than $n + 4 \cdot \log_2(1/\delta)$ coins, where one factor of 2 is due to the use of expanders and the other to the "median-of-averages paradigm". As long as we stick to using expanders in the Median-of-Averages Sampler, there is no hope to reduce the first factor, which is due to the relation between the expander degree and its second eigenvalue. In fact, achieving a factor of 4 rather than a bigger factor is due to the use of Ramanujan Graphs (which have the best possible such relation).

[7] Partial support for this conjecture was offered to us recently by Ronen Shaltiel (priv. comm., 2010). He observed that one $\log_2(1/\varepsilon)$ term can be shaved off the lower bound in the special case of averaging samplers (see below), by using the connection to randomness extractors and a lower bound on entropy loss due to [88].

lower bound (even for Boolean)	$n + \log_2(1/\delta) - 2\log_2(1/\varepsilon) - \log_2\log_2(1/\delta) - O(1)$
upper bound	$n + \log_2(1/\delta) - \log_2\log_2(1/\delta)$
efficient samplers	$n + (4+\alpha)\log_2(1/\delta) + \log_2(1/\varepsilon)$, for any $\alpha > 0$
efficient Boolean samplers	$n + (4+\alpha)\log_2(1/\delta)$, for any $\alpha > 0$

Figure 6.2: The randomness complexity of samplers that make $\Theta(\frac{\log(1/\delta)}{\varepsilon^2})$ queries.

Boolean samplers vs general ones. Another fact presented in Figure 6.2 is that we can currently do better if we are guaranteed that the oracle function is Boolean (rather than mapping to the interval $[0,1]$). We stress that the lower bound holds also with respect to samplers that need only to work for Boolean functions.

Adaptive vs non-adaptive. All known samplers are non-adaptive; that it, they determine the sample points (queries) solely as a function of their coin tosses. In contrast, *adaptive* samplers may determine the next query depending on the value of the function on previous queries. Intuitively, adaptivity should not help the sampler. Indeed, all lower bounds refer also to adaptive samplers, whereas all upper bound only utilizes non-adaptive samplers. This indicates that the difference between adaptive samplers and non-adaptive ones can not be significant. In a preliminary version of this survey we advocated providing a direct and more tight proof of the foregoing intuition. When referring to the sample complexity, such a simple proof was provided in [15, Lem. 9]: It amounts to observing that adapting queries made to a random isomorphic copy of a function f are equivalent to uniformly and independently distributed queries made to f. Thus, adaptivity offers no advantage in this setting.

Averaging (or oblivious) **samplers.** A special type of non-adaptive samplers are ones that output the average value of the function over their sample points. Such samplers were first defined in [20], where they were called "oblivious", but we prefer the term *averaging*. (Recall that we have already defined and used such samplers in Section 6.6.3.) We mention that averaging samplers have some applications not offered by arbitrary non-adaptive samplers (cf., [20] and [103]). More importantly, averaging samplers are very appealing, since averaging over a sample seem *the natural thing to do*. Furthermore, as pointed out in [117], averaging samplers are closely related to randomness extractors (see Section 6.8 and more details in [94]). Note that the Naive Sampler, the Pairwise-Independent Sampler, and the Expander Sampler are all averaging samplers, although they differ in the way they generate their sample. However, the Median-of-Averages Sampler, as its name indicates, is not an averaging sampler. Thus, obtaining an averaging sampler of relatively low sample and randomness complexities requires an alternative approach. The best results are obtained via the connection to randomness extractors, and are summarized below.

Theorem 6.20 (efficient averaging samplers [91, Cor. 7.3]):[8] *For every constant $\alpha > 0$, there exists an efficient* averaging *sampler with*

[8] The result builds on [117], and uses [55, Thm. 1.5] in order to remove a mild restriction on the value of ε.

- Sample Complexity: $\mathrm{poly}(\varepsilon^{-1}, \log(1/\delta))$.
- Randomness Complexity: $n + (1+\alpha) \cdot \log_2(1/\delta)$.

We stress that this sampler is not sample-optimal (i.e., the polynomial in ε^{-1} is not quadratic). It would be interesting to obtain an efficient sample-optimal *averaging* sampler of low randomness complexity, say, one that uses $O(n + \log(1/\delta))$ coins. We mention that non-explicit sample-optimal averaging samplers of low randomness complexity do exist; specifically, Theorems 6.4 and 6.5 holds with averaging-samplers (see [27, 117], resp.).

6.8 Advanced Topic: A Different Perspective

As stated in the introduction, the intention of the current survey was to provide a wide audience of theoretical computer scientists with a basic tutorial regarding samplers. The focus of this tutorial was on the complexity of sampling, and our aim was to *simultaneously* minimize three complexity measures: (1) the sample complexity, (2) the randomness complexity, and (3) the computational complexity. We actually focused on the minimization of the first two, while requiring that a minimal level of computational efficiency is maintained (i.e., that the sampler works in time that is polynomial in the total length of the queries made).

From our perspective, averaging samplers are of no special interest, except maybe for their natural appeal. An alternative perspective, strongly advocated by Ronen Shaltiel and Amnon Ta-Shma, may put averaging samplers and their relation to general samplers at the main focus. This is likely to yield a very interesting survey, which we outline in the rest of this section, but it is not the one we set out to write...

6.8.1 Averaging Samplers versus General Samplers

The alternative survey will focus on the question of whether non-averaging samplers can outperform averaging samplers. As noted by Amnon and Ronen, a good starting point for such a survey is the observation that the median of averages operation can be used for improving the performance of samplers, but it yields non-averaging samplers. Specifically, the median of averages operation can be combined with simple averaging samplers (e.g., the pairwise independent ones) to yield very strong and simple non-averaging samplers. Another interesting observation is that the currently known lower bound on the randomness complexity of sample-optimal averaging samplers is higher than the currently know bound for general samplers (see Footnote 7). Finally, when viewing the minimization of sample complexity as the primary goal and the minimization of the randomness complexity as the secondary goal, the median of averages operation enables constructing *efficient* samplers that are by far better (and also much simpler) than the currently known *efficient* averaging samplers.

Another interesting parameter is the Boolean versus general distinction, which was discussed in prior sections. Recall that in the case of averaging samplers, the two notions are almost identical (see Theorem 6.19), whereas for general sampler we currently lose a $\log_2(1/\varepsilon)$ term in the randomness complexity (see Theorem 6.14). Focusing on sample-optimal samplers, we summarize the currently known results in Figure 6.3, where the three

lower bound (even for Boolean)	$n + \log_2(1/\delta) - 2\log_2(1/\varepsilon) - \ell - O(1)$
lower bound for averaging samplers	$n + \log_2(1/\delta) - \log_2(1/\varepsilon) - \ell - O(1)$
upper bound (by averaging samplers)	$n + \log_2(1/\delta) - \ell$
efficient samplers	$n + (4+\alpha)\cdot\log_2(1/\delta) + \log_2(1/\varepsilon)$, $\forall\alpha > 0$
efficient averaging samplers	$n + (1+\alpha)\cdot\log_2(1/\delta) + \widetilde{O}(s)$, $\forall\alpha > 0$

Figure 6.3: The randomness complexity of samplers that make $s \stackrel{\text{def}}{=} \Theta(\frac{\log(1/\delta)}{\varepsilon^2})$ queries, where ℓ denotes $\log_2\log_2(1/\delta)$.

first rows ignore the question of efficiency (and the last row of Figure 6.3 is justified by combining Theorems 6.20 and 6.17).[9]

6.8.2 Averaging Samplers versus Randomness Extractors

We start by recalling the basic definition of randomness extractors, while (slightly) changing some common conventions to better fit our discussion.[10] Loosely speaking, a randomness extractor is a function $\text{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$ that uses an $(\log_2 s)$-bit long random seed in order to transform an r-bit long (outcome of a) weak source of randomness into an n-bit long string that is almost uniformly distributed in $\{0,1\}^n$. Specifically, we consider arbitrary weak sources that are restricted (only) in the sense that, for a parameter k, no string appears as the source outcome with probability that exceeds 2^{-k}. Such sources are called (r,k)-sources (and k is called the min-entropy). A special type of (r,k)-sources are (r,k)-flat sources, which are sources in which each string appears with probability that equals either 2^{-k} or 0. We say that two distributions are ϵ-close if the statistical difference (a.k.a variation distance) between them is at most ϵ. Now, Ext is called a (k,ϵ)-extractor if for any (r,k)-source X it holds that $\text{Ext}(X, U_s)$ is ϵ-close to the uniform distribution over n-bit strings, where U_s denotes the uniform distribution over $[s]$.

There is a close relationship between extractors and averaging samplers. In order to discuss this relationship, it will be more convenient to state the performance guarantees of the sampler (i.e., ε and δ) in terms of its complexities (i.e., s and r), rather than the other way around (as done in the rest of this survey). Thus, we may say that a certain oracle machine (which has certain sample and randomness complexities) is an (ε,δ)-sampler if it satisfies Eq. (6.1) for these particular values of ε and δ.

We shall first show that any averaging sampler gives rise to an extractor. Let $G : \{0,1\}^r \to (\{0,1\}^n)^s$ be the sample generating algorithm of an averaging (ε,δ)-sampler. That is, G uses r bits of randomness and generates s sample points in $\{0,1\}^n$ such that, for every $f : \{0,1\}^n \to [0,1]$ with probability at least $1-\delta$, the average of the f-values of these s pseudorandom points resides in the interval $[\overline{f} \pm \varepsilon]$, where $\overline{f} \stackrel{\text{def}}{=} \sum_{x\in\{0,1\}^n} f(x)/2^n$. Define $\text{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$ such that $\text{Ext}(\omega, i)$ is the i^{th} sample generated by $G(\omega)$. We shall prove that Ext is a $(k, 2\varepsilon)$-extractor, for $k = r - \log_2(\varepsilon/\delta)$.

[9] Specifically, we invoke Theorem 6.17 when using the sampler of Theorem 6.20 as the first (i.e., "outer") sampler, and the Naive Sampler as the second (i.e., "inner") sampler.

[10] Typically, extractors are defined as mapping $\{0,1\}^n \times \{0,1\}^s$ to $\{0,1\}^m$.

Suppose towards the contradiction that there exists a (r,k)-source X such that for some $S \subset \{0,1\}^n$ it is the case that $\mathsf{Pr}[\mathrm{Ext}(X,U_s) \in S] > 2^{-n} \cdot |S| + 2\varepsilon$. Then, without loss of generality (see Exercise 8.1), X is (r,k)-flat, and we consider the set

$$B = \{x \in \{0,1\}^r : \mathsf{Pr}[\mathrm{Ext}(x,U_s) \in S] > 2^{-n} \cdot |S| + \varepsilon\}.$$

Then, $|B| > \varepsilon \cdot 2^k = \delta \cdot 2^r$, where the inequality holds since $\mathsf{Pr}[\mathrm{Ext}(X,U_s) \in S] \leq \mathsf{Pr}[X \in B] + 2^{-n} \cdot |S| + \varepsilon$. Defining $f(z) = 1$ if $z \in S$ and $f(z) = 0$ otherwise, it holds that $\overline{f} = |S|/2^m$. But, for every $\omega \in B$, the f-average of the sample $G(\omega)$ is greater than $\overline{f} + \varepsilon$, in contradiction to the hypothesis that the sampler has error probability δ (with respect to accuracy ε).

We now turn to show that extractors give rise to averaging samplers. Let $\mathrm{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$ be a (k,ϵ)-extractor. Consider the sample generation algorithm $G : \{0,1\}^r \to (\{0,1\}^n)^s$ defined by $G(\omega) = (\mathrm{Ext}(\omega,i))_{i\in[s]}$. We prove that G corresponds to an averaging (ε,δ)-sampler, for $\delta = 2^{-(r-k-1)}$.

Suppose towards the contradiction that there exists a function $f : \{0,1\}^n \to [0,1]$ such that for $\delta 2^r = 2^{k+1}$ strings $\omega \in \{0,1\}^r$ the average f-value of the sample $G(\omega)$ deviates from $\overline{f}$ by more than ϵ. Suppose, without loss of generality, that for at least half of these ω's the average is greater than $\overline{f} + \epsilon$, and let B denote the set of these ω's. Then, for X that is uniformly distributed on B (and is thus a (r,k)-source), we have

$$\mathsf{E}[f(\mathrm{Ext}(X,U_s))] > \mathsf{E}[f(U'_n)] + \varepsilon,$$

where U'_n denotes the uniform distribution on n-bit long strings. But, since $|f(z)| \leq 1$ for every z, this contradicts the hypothesis that $\mathrm{Ext}(X,U_s)$ is ε-close to U'_n, because $|\mathsf{E}[f(Y)] - \mathsf{E}[f(Z)]|$ is upper bounded by the statistical difference between Y and Z (times $\max_z\{|f(z)|\}$). Summarizing the foregoing discussion, we obtain:

Theorem 6.21 (averaging samplers vs randomness extractors): *Let $r,s,k \in \mathbb{N}$ and $\varepsilon,\delta \in [0,1]$. Then:*

1. *If $\mathrm{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$ is a (k,ε)-extractor, then the sample generating algorithm $G : \{0,1\}^r \to (\{0,1\}^n)^s$ defined by $G(\omega) = (\mathrm{Ext}(\omega,i))_{i\in[s]}$ yields an averaging (ε,δ)-sampler for $\delta = 2^{-(r-k-1)}$* (i.e., $r - k = \log_2(1/\delta) + 1$).

2. *If $G : \{0,1\}^r \to (\{0,1\}^n)^s$ is the sample generating algorithm of an averaging (ε,δ)-sampler, then the algorithm $\mathrm{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$ defined by $\mathrm{Ext}(\omega,i) = G(\omega)_i$ is a $(k,2\varepsilon)$-extractor, for $k = r - \log_2(\varepsilon/\delta)$* (i.e., $r - k = \log_2(1/\delta) - \log_2(1/\varepsilon)$).

Note that starting with a $(k,2\varepsilon)$-extractor and applying both parts of Theorem 6.21, we obtain a $(k',2\varepsilon)$-extractor for $k' = k + 1 + \log_2(1/\varepsilon)$. Thus, the translation offered by Theorem 6.21 is not optimal, yet the bounds provided in both directions are (in general) tight.[11]

[11] To see the tightness of Part 1, consider an arbitrary (k,ε)-extractor, $\mathrm{Ext} : \{0,1\}^r \times [s] \to \{0,1\}^n$, and modify it such that, for every $x' \in \{0,1\}^k$ and $i \in [3\varepsilon \cdot s]$, it holds that $\mathrm{Ext}(0^{r-k}x',i) = 0^n$. Then, the modified extractor is a $(k+2,2\varepsilon)$-extractor, but the resulting averaging sampler has error probability *at least* 2^{-r+k} with respect to deviation 2ε. (Recall that Part 1 asserts that the resulting averaging sampler

The connection to averaging samplers and the desire to have averaging samplers of optimal sample and randomness complexities calls attention to a research direction regarding extractors that did not receive much attention. We refer to the construction of extractors with strongly optimal seed length and almost optimal extraction rate. That is, the seed length, which is $\log_2 s$ in terms of this section, should be optimal up to a *constant additive term*, whereas the extraction rate (i.e., n/k) (or rather the inverse loss rate (i.e., $(r-k)/(n-k)$)) should be close to 1.

6.9 Perspective: The Hitting problem

The hitting problem is a one-sided version of the Boolean sampling problem. Given parameters n (length), ε (density) and δ (error), and oracle access to any function $\sigma : \{0,1\}^n \to \{0,1\}$ such that $|\{x : f(x)=1\}| \geq \varepsilon 2^n$, the task is to find a string that is mapped to 1. That is:

Definition 6.22 (hitter): *A* hitter *is a randomized algorithm that on input parameters n, ε and δ, and oracle access to* any *function $\sigma:\{0,1\}^n\to\{0,1\}$, such that $|\{x : f(x)=1\}| \geq \varepsilon 2^n$, satisfies*

$$\mathsf{Pr}[\sigma(\mathrm{hitter}^{\sigma}(n,\varepsilon,\delta)) = 1] \;>\; 1-\delta.$$

Observe that, on input parameters n, ε and δ, any sampler must be able to distinguish the all-zero function from any function $\sigma : \{0,1\}^n \to \{0,1\}$ such that $|\{x : f(x) = 1\}| \geq 2\varepsilon 2^n$. Thus, in the latter case, the sampler must obtain (with probability at least $1-\delta$) the value 1 for at least one of its queries, and outputting such a query satisfies the requirement for a hitter (w.r.t parameters n, 2ε and δ).

We note that all results and techniques regarding sampling (presented in the main text), have simpler analogous with respect to the hitting problem. In fact, this appendix may be read as a warm-up towards the main text.

6.9.1 The Information Theoretic Perspective

Analogously to the Naive Sampler, we have the Naive Hitter that *independently* selects $m \stackrel{\text{def}}{=} \frac{\ln(1/\delta)}{\varepsilon}$ uniformly distributed sample points and queries the oracle on each. Clearly, the probability that the hitter fails to sample a point of value 1 is at most $(1-\varepsilon)^m = \delta$. The complexities of this hitter are as follows

- Sample Complexity: $m \stackrel{\text{def}}{=} \frac{\ln(1/\delta)}{\varepsilon} = \Theta(\frac{log(1/\delta)}{\varepsilon})$.

has error probability at most $2^{-(r-k-3)}$ with respect to deviation 2ε.) To see the tightness of Part 2, consider an arbitrary avearging (ε,δ)-sampler with a sample generating algorithm $G : \{0,1\}^r \to (\{0,1\}^n)^s$, and modify the latter to be identically zero on $\delta 2^r$ seeds; that is, for an arbitrary $B \subset \{0,1\}^r$ of size $\delta 2^r$, redefine G such that for every $x \in B$ it holds that $G(x) = (0^n)^s$. Then, the modified averaging sampler is an $(\varepsilon, 2\delta)$-sampler, but (as shown next) the resulting extractor can be a $(k', c\varepsilon)$-extractor only if $k' > k + \log_2(1/\varepsilon) - c'$, where $k \stackrel{\text{def}}{=} r - \log_2(1/\delta)$ and $c' = \log_2(c+1)$. The lower bound on k' holds because a (k', r)-source may assign B probability $2^{k-k'}$, whereas 0^n should be assigned probability at most $c\varepsilon + 2^{-n}$. Thus, $2^{k-k'} \leq c\varepsilon + 2^{-n}$, which implies $k' - k > \log_2(1/\varepsilon) - c'$. (Recall that Part 2 asserets that the resulting construct is a $(k', 2\varepsilon)$-extractor for $k' = k + \log_2(1/\varepsilon) + 1$.)

- Randomness Complexity: $m \cdot n = \Theta(\frac{log(1/\delta)}{\varepsilon} \cdot n)$.
- Computational Complexity: Indeed efficient.

It is easy to prove that the Naive Hitter is sample-optimal. That is:

Theorem 6.23 (sample complexity lower bound): *Any hitter has sample complexity bounded below by*

$$\min\left\{2^{n-O(1)}, \frac{\ln(1/2\delta)}{2\varepsilon}\right\}$$

provided $\varepsilon \leq \frac{1}{8}$.

Proof Sketch: Let A be a hitter with sample complexity $m = m(n, \varepsilon, \delta)$ and let σ be a function selected at random by setting its value independently on each argument such that $\mathsf{Pr}(\sigma(x)=1) = 1.5\varepsilon$. Then,

$$\mathsf{Pr}_\sigma[\sigma(A^\sigma(n, \varepsilon, \delta)) \neq 1] \;=\; (1 - 1.5\varepsilon)^m,$$

where the probability is taken over the choice of σ and the internal coin tosses of A. On the other hand, using a Multiplicative Chernoff Bound:

$$\mathsf{Pr}_\sigma[|\{x : \sigma(x)=1\}| < \varepsilon 2^n] = 2\exp(-\Omega(\varepsilon 2^n)).$$

We may assume that $\Omega(\varepsilon 2^n) > \log_2(1/\delta)$ and so the probability that σ has at least ε fraction of 1's and yet algorithm A fails is at least $(1-1.5\varepsilon)^m - \delta > \delta$, unless $m > \frac{\ln(1/2\delta)}{\ln(1-1.5\varepsilon)} > \frac{\ln(1/2\delta)}{2\varepsilon}$. ■

Theorem 6.24 (randomness complexity lower bound): *Let* $s : \mathbb{N} \times [0,1]^2 \to \mathbb{R}$. *Any sampler that has sample complexity at most* $s(n, \varepsilon, \delta)$*, has randomness complexity at least*

$$r > n - \log_2 s(n, \varepsilon, \delta) + \log_2((1 - \varepsilon)/\delta).$$

Proof Sketch: Let A be a hitter with sample complexity $s = s(n, \varepsilon, \delta)$, and randomness complexity $r = r(n, \varepsilon, \delta)$. Consider any subset of $\delta 2^r$ possible sequence of coin tosses for A and all $\delta 2^r \cdot s$ points that are queried at any of these coin-sequences. We argue that $\delta 2^r \cdot s > (1 - \varepsilon)2^n$ must hold, or else there exists a function σ that evaluates to 0 on each of these points and to 1 otherwise (contradicting the requirement that this function be "hit" with probability at least $1 - \delta$). Thus, $r > n + \log_2(1 - \varepsilon) - \log_2 s + \log_2(1/\delta)$. ■

6.9.2 The Pairwise-Independent Hitter

Using a pairwise-independent sequence of uniformly distributed sample points rather than a totally independent one, we obtain the pairwise-independent hitter. Here we set $m \stackrel{\text{def}}{=} \frac{1-\varepsilon}{\delta\varepsilon}$.

Letting ζ_i represent the σ-value of the i^{th} sample point, considering only σ's with an ε-fraction of 1-values,[12] and using Chebyshev's Inequality we have

$$\begin{aligned}\mathsf{Pr}\left[\sum_{i=1}^{m}\zeta_i = 0\right] &\leq \mathsf{Pr}\left[\left|m\varepsilon - \sum_{i=1}^{m}\zeta_i\right| \geq \varepsilon m\right] \\ &\leq \frac{m\cdot(1-\varepsilon)\varepsilon}{(\varepsilon m)^2} \\ &= \delta.\end{aligned}$$

Recalling that we can generate $2^n - 1$ pairwise-independent samples using $2n$ coins, the pairwise-independent hitter achieves

- Sample Complexity: $\frac{1}{\delta\varepsilon}$ (reasonable for constant δ).
- Randomness Complexity: $2n$
- Computational Complexity: Indeed efficient.

6.9.3 The combined Hitter

Our goal here is to decrease the sample complexity of the Pairwise-Independent Hitter while essentially maintaining its randomness complexity. To motivate the new construction we first consider an oversimplified version of it.

Combined Hitter (oversimplified): On input parameters n, ε and δ, set $m \stackrel{\text{def}}{=} \frac{2}{\varepsilon}$ and $\ell \stackrel{\text{def}}{=} \log_2(1/\delta)$, generate ℓ independent m-element sequences, each being a sequence of m *pairwise-independently and uniformly distributed* strings in $\{0,1\}^n$. Denote the sample points in the i^{th} sequence by $s^i_1, ..., s^i_m$. We merely try all these $\ell \cdot m$ samples as hitting points. Clearly, for each $i = 1, ..., \ell$,

$$\mathsf{Pr}[(\forall j \in \{1, .., m\})\ \sigma(s^i_j) = 0] < \frac{1}{2}$$

and so the probability that none of these s^i_j "hits σ" is at most $0.5^\ell = \delta$. Thus, the oversimplified version described above is indeed a hitter and has the following complexities:

- Sample Complexity: $\ell \cdot m = O(\frac{\log(1/\delta)}{\varepsilon})$.
- Randomness Complexity: $\ell \cdot O(n) = O(n \cdot \log(1/\delta))$.
- Computational Complexity: Indeed efficient.

[12] Considering only σ's with *exactly* an ε-fraction of 1-values implies that $\mathsf{Var}[\zeta_i] = (1-\varepsilon)\varepsilon$. Needless to say, if the hitter works well for all these functions, then it works well for all functions having *at least* an ε-fraction of 1-values.

Thus, the sample complexity is optimal (upto a constant factor), but the randomness complexity is higher than what we aim for. To reduce the randomness complexity, we use the same approach as above, but take dependent sequences rather than independent ones. The dependency we use is such that essentially preserves the probabilistic behavior of independent choices. Specifically, we use random walks on expander graphs (cf. Lecture 5) to generate a sequence of ℓ "seeds" each of length $O(n)$. Each seed is used to generate a sequence of m pairwise independent elements in $\{0,1\}^n$, as above. Thus, we obtain:

Corollary 6.25 (The Combined Hitter): *There exists an efficient hitter with*

- Sample Complexity: $O(\frac{\log(1/\delta)}{\varepsilon})$.
- Randomness Complexity: $2n + O(\log(1/\delta))$.

Furthermore, we can obtain randomness complexity $2n + (2 + o(1)) \cdot \log_2(1/\delta))$.

Proof Sketch: We use an explicit construction of expander graphs with vertex set $\{0,1\}^{2n}$, degree d and second eigenvalue λ so that $\lambda/d < 0.1$. We consider a random walk of (edge) length $\ell - 1 = \log_2(1/\delta)$ on this expander, and use each of the ℓ vertices along the path as random coins for the Pairwise-Independent Hitter, which in turn makes $m \stackrel{\text{def}}{=} \varepsilon/3$ trials. To analyze the performance of the resulting algorithm, we let W denote the set of coin tosses (for the basic hitter) on which the basic hitter fails to output a point that evaluates to 1. By the hypothesis, $\frac{|W|}{2^{2n}} \leq 1/3$, and using (Part 1) of Lemma 5.5, the probability that all vertices of a random path reside in W is bounded above by $(0.34 + 0.1)^\ell < \delta$. The furthermore clause follows by using a Ramanujan Graph and an argument as in the proof of Item 2 of Theorem 6.8. ■

6.9.4 The Expander Hitter

Our goal here is to decrease the randomness complexity of hitters from $2n + O(\log(1/\delta))$ to $n + O(\log(1/\delta))$, while preserving the sample complexity of $O(\varepsilon^{-1} \log(1/\delta))$. The first step is to get an analogous improvement with respect to the Pairwise-Independent Hitter (which has sample complexity $O(1/\delta\varepsilon)$).

We use a Ramanujan Graph of degree $d = O(1/\varepsilon\delta)$ and vertex-set $\{0,1\}^n$. The hitter uniformly selects a vertex in the graph and use its neighbors as a sample. Suppose we try to hit a 1-value of a function σ and let $S \stackrel{\text{def}}{=} \{u : \sigma(u) = 1\}$. Let $B \stackrel{\text{def}}{=} \{v : \mathrm{N}(v) \cap S = \emptyset\}$ be the set of bad vertices (i.e., choosing any of these results in not finding a preimage of 1). Using the Expander Mixing Lemma we have

$$\begin{aligned} \rho(B)\rho(S) &= \left| \frac{|(B \times S) \cap E|}{|E|} - \rho(B)\rho(S) \right| \\ &\leq \frac{\lambda}{d} \cdot \sqrt{\rho(B)\rho(S)} \end{aligned}$$

Hence, $\rho(B)\rho(S) \leq (\lambda/d)^2 = \varepsilon\delta$ and using $\rho(S) \geq \varepsilon$ we get $\rho(B) \leq \delta$. The complexities of this hitter are as follows:

- Sample Complexity: $O(\frac{1}{\delta\varepsilon})$
- Randomness Complexity: n
- Computational Complexity: Indeed efficient.

Adapting the argument in the proof of Corollary 6.25, we obtain

Corollary 6.26 (The Combined Hitter, revisited): *There exists an efficient hitter with*

- Sample Complexity: $O(\frac{\log(1/\delta)}{\varepsilon})$.
- Randomness Complexity: $n + (2 + o(1)) \cdot \log_2(1/\delta))$.

Lecture 7

Approximate Counting and Uniform Generation

This lecture differs from other lectures in this volume in paying more attention to the application of the randomized method, and actually in starting from the application and getting to the method later. The methods that we shall introduce and use are (1) the use of hashing as a "random seiving" procedure, and (2)-the close relationship between approximate counting and uniform generation. Let us start with a fast overview of these two methods.

Random sieving refers to a situation in which we wish to pinpoint few elements of a large set, say $S \subset \{0,1\}^n$. If we know (or approximately guess) $|S|$, then we can use hashing to create a random sieve that allows elements to pass with probability $1/|S|$. Thus, we can expect few elements of S to pass this sieve. This idea is used in Sections 7.2.2, 7.3, and 7.4.2. In Section 7.4.1 we explore the relationship between approximating the size of sets and uniformly selecting elements in them. We show that in many settings (i.e., when the approximations and uniform generation hold also for some natural subsets of these sets), these tasks are computationally equivalent.

Indeed, when referring to approximation or counting problems, we refer to counting objects that can be efficiently recognized. The two formulations of NP provide a suitable definition of such objects and yield corresponding counting problems:

1. Counting the number of solutions for a given instance of a search problem (of a relation) $R \subseteq \{0,1\}^* \times \{0,1\}^*$ having efficiently checkable solutions. That is, on input x, we are required to output $|\{y : (x,y) \in R\}|$.

2. Counting the number of NP-witnesses (with respect to a specific verification procedure V) for a given instance of an NP-set S (i.e., $S \in \mathcal{NP}$ and V is the corresponding verification procedure). That is, on input x, we are required to output $|\{y : V(x,y) = 1\}|$.

We shall consider these counting problems as well as relaxations of them (which refer to approximating the said quantities), and see connections between these relaxed counting problems and randomized algorithms.

Indeed, our general context is that of NP-search problems. Specifically, we consider the class of search problems having efficiently checkable soltions. Formally, we denote this class by $\mathcal{PC}$ (standing for "Polynomial-time Check")[1], and define it as the class of search problems that correspond to polynomially-bounded binary relations that have efficiently checkable solutions. That is, $R \in \mathcal{PC}$ if the following two conditions hold:

1. For some polynomial p, if $(x,y) \in R$ then $|y| \leq p(|x|)$.
2. There exists a polynomial-time algorithm that given (x,y) determines whether or not $(x,y) \in R$.

Indeed, searching for NP-witnesses (w.r.t some set in $\mathcal{NP}$) is a special case. However, here we are not interrested in the task of finding solutions for such NP-type search problems or determining whether such exists but rather in two related computational problems (which vastly generalized the former): (1) finding a uniformly distributed solution and (2) counting (or approximating) the number of solutions.

Although our focus is on the problem of approximate counting and its relation to uniform generation, we start by recalling the basic facts regarding exact counting.

7.1 Background: Exact Counting

In continuation to the foregoing discussion, we define the class of problems concerned with counting efficiently recognized objects. (Recall that $\mathcal{PC}$ denotes the class of search problems having polynomially long solutions that are efficiently checkable.)

Definition 7.1 (counting efficiently recognized objects – $\#\mathcal{P}$): *The class $\#\mathcal{P}$ consists of all functions that count solutions to a search problem in $\mathcal{PC}$. That is, $f : \{0,1\}^* \to \mathbb{N}$ is in $\#\mathcal{P}$ if there exists $R \in \mathcal{PC}$ such that, for every x, it holds that $f(x) = |R(x)|$, where $R(x) = \{y : (x,y) \in R\}$. In this case we say that f is the* counting problem associated with *R, and denote the latter by $\#R$* (i.e., *$\#R = f$).*

Every decision problem in $\mathcal{NP}$ is Cook-reducible to $\#\mathcal{P}$, because every such problem can be cast as deciding membership in $S_R = \{x : |R(x)| > 0\}$ for some $R \in \mathcal{PC}$. It also holds that $\mathcal{BPP}$ is Cook-reducible to $\#\mathcal{P}$. The class $\#\mathcal{P}$ is sometimes defined in terms of decision problems, as is implicit in the following proposition.

Proposition 7.2 (a decisional version of $\#\mathcal{P}$): *For any $f \in \#\mathcal{P}$, deciding membership in $S_f \stackrel{\text{def}}{=} \{(x,N) : f(x) \geq N\}$ is computationally equivalent to computing f.*

Actually, the claim holds for any function $f : \{0,1\}^* \to \mathbb{N}$ for which there exists a polynomial p such that for every $x \in \{0,1\}^*$ it holds that $f(x) \leq 2^{p(|x|)}$.

Proof: Since the relation R vouching for $f \in \#\mathcal{P}$ (i.e., $f(x) = |R(x)|$) is polynomially bounded, there exists a polynomial p such that for every x it holds that $f(x) \leq 2^{p(|x|)}$. Deciding membership in S_f is easily reduced to computing f (i.e., we accept the input (x,N) if and only if $f(x) \geq N$). Computing f is reducible to deciding S_f by using a binary

[1] The notation $\mathcal{PC}$ was advocated in [44], and is intended to replace the notation $\mathcal{FNP}$ used elsewhere.

search. This relies on the fact that, on input x and oracle access to S_f, we can determine whether or not $f(x) \geq N$ by making the query (x, N). Note that we know a priori that $f(x) \in [0, 2^{p(|x|)}]$. ■

The counting class $\#\mathcal{P}$ is also related to the problem of enumerating all possible solutions to a given instance (see Exercise 7.1).

7.1.1 On the power of $\#\mathcal{P}$

As indicated, $\mathcal{NP} \cup \mathcal{BPP}$ is (easily) reducible to $\#\mathcal{P}$. Furthermore, as stated in Theorem 7.3, the entire Polynomial-Time Hierarchy is Cook-reducible to $\#\mathcal{P}$ (i.e., $\mathcal{PH} \subseteq \mathcal{P}^{\#\mathcal{P}}$). On the other hand, any problem in $\#\mathcal{P}$ is solvable in polynomial space, and so $\mathcal{P}^{\#\mathcal{P}} \subseteq \mathcal{PSPACE}$.

Theorem 7.3 [102] *Every set in $\mathcal{PH}$ is Cook-reducible to $\#\mathcal{P}$.*

We do not present a proof of Theorem 7.3 here, because the known proofs are rather technical. Furthermore, one main idea underlying these proofs appears in a more clear form in the proof of Theorem 7.16.

7.1.2 Completeness in $\#\mathcal{P}$

The definition of $\#\mathcal{P}$-completeness is analogous to the definition of $\mathcal{NP}$-completeness. That is, *a counting problem f is $\#\mathcal{P}$-*complete *if $f \in \#\mathcal{P}$ and every problem in $\#\mathcal{P}$ is Cook-reducible to f.*

We claim that the counting problems associated with the NP-complete problems presented in previous lectures are all $\#\mathcal{P}$-complete. We warn that this fact is not due to the mere NP-completeness of these problems, but rather to an additional property of the reductions establishing their NP-completeness. Specifically, the Karp-reductions that were used (or variants of them) have the extra property of preserving the number of NP-witnesses (as captured by the following definition).

Definition 7.4 (parsimonious reductions): *Let $R, R' \in \mathcal{PC}$ and let g be a Karp-reduction of $S_R = \{x : R(x) \neq \emptyset\}$ to $S_{R'} = \{x : R'(x) \neq \emptyset\}$, where $R(x) = \{y : (x, y) \in R\}$ and $R'(x) = \{y : (x, y) \in R'\}$. We say that g is* parsimonious (with respect to R and R') *if for every x it holds that $|R(x)| = |R'(g(x))|$. In such a case we say that g is a* parsimonious reduction of R to R'.

We stress that the condition of being parsimonious refers to the two underlying relations R and R' (and not merely to the sets S_R and $S_{R'}$). The requirement that g is a Karp-reduction is partially redundant, because if g is polynomial-time computable and for every x it holds that $|R(x)| = |R'(g(x))|$, then g constitutes a Karp-reduction of S_R to $S_{R'}$. Specifically, $|R(x)| = |R'(g(x))|$ implies that $|R(x)| > 0$ (i.e., $x \in S_R$) if and only if $|R'(g(x))| > 0$ (i.e., $g(x) \in S_{R'}$). The reader may easily verify that the Karp-reduction underlying the proof of CSAT (and SAT) as well as many of the reductions used in the theory of NP-completeness are parsimonious.

Theorem 7.5 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then the counting problem associated with R is $\#\mathcal{P}$-complete.*

Proof: Clearly, the counting problem associated with R, denoted $\#R$, is in $\#\mathcal{P}$. To show that every $f' \in \#\mathcal{P}$ is reducible to f, we consider the relation $R' \in \mathcal{PC}$ that is counted by f'; that is, $\#R' = f'$. Then, by the hypothesis, there exists a parsimonious reduction g of R' to R. This reduction also reduces $\#R'$ to $\#R$; specifically, $\#R'(x) = \#R(g(x))$ for every x. ■

Corollaries. As an immediate corollary of Theorem 7.5, we get that counting the number of satisfying assignments to a given CNF formula is $\#\mathcal{P}$-complete. Similar statement hold for all the other NP-complete problems mentioned in previous lectures and in fact for all NP-complete problems listed in [40]. These corollaries follow from the fact that all known reductions among natural NP-complete problems are either parsimonious or can be easily modified to be so.

We conclude that many counting problems associated with NP-complete search problems are $\#\mathcal{P}$-complete. It turns out that also counting problems associated with efficiently solvable search problems may be $\#\mathcal{P}$-complete.

Theorem 7.6 *There exist $\#\mathcal{P}$-complete counting problems that are associated with efficiently solvable search problems. That is, there exists $R \in \mathcal{PF}$* (i.e., R is solvable in polynomial-time) *such that $\#R$ is $\#\mathcal{P}$-complete.*

Proof: Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ, τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Note that the search problem of $R_{\mathtt{dnf}}$ is easy to solve (e.g., by picking an arbitrary truth assignment that satisfies the first term in the input formula). To see that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete consider the following reduction from $\#R_{\mathtt{SAT}}$ (which is $\#\mathcal{P}$-complete by Theorem 7.5). Given a CNF formula ϕ, transform $\neg\phi$ into a DNF formula ϕ' by applying de-Morgan's Law, and return $2^n - \#R_{\mathtt{dnf}}(\phi')$, where n denotes the number of variables in ϕ (resp., ϕ'). ■

Reflections. We note that Theorem 7.6 is not established by a parsimonious reduction. This fact should not come as a surprise because a parsimonious reduction of $\#R'$ to $\#R$ implies that $S_{R'} = \{x : \exists y \text{ s.t. } (x,y) \in R'\}$ is reducible to $S_R = \{x : \exists y \text{ s.t. } (x,y) \in R\}$, where in our case $S_{R'}$ is NP-Complete while $S_R \in \mathcal{P}$ (since $R \in \mathcal{PF}$). Nevertheless, the proof of Theorem 7.6 is related to the hardness of some underlying decision problem (i.e., the problem of deciding whether a given DNF formula is a tautology (i.e., whether $\#R_{\mathtt{dnf}}(\phi') = 2^n$)). But does there exist a $\#\mathcal{P}$-complete problem that is "not based on some underlying NP-complete decision problem"? Amazingly enough, the answer is positive.

Theorem 7.7 [106] *Counting the number of perfect matchings in a bipartite graph is $\#\mathcal{P}$-complete.*

Equivalently (see Exercise 7.2), the problem of computing the permanent of matrices with 0/1-entries is $\#\mathcal{P}$-complete. Recall that the permanent of an n-by-n matrix $M = (m_{i,j})$, denoted $\mathtt{perm}(M)$, equals the sum over all permutations π of $[n]$ of the products $\prod_{i=1}^{n} m_{i,\pi(i)}$. Theorem 7.7 is proven by composing the following two (many-to-one) reductions (asserted in Propositions 7.8 and 7.9, respectively) and using the fact that $\#R_{\mathtt{3SAT}}$ is $\#\mathcal{P}$-complete (see Theorem 7.5). Needless to say, the resulting reduction is not parsimonious.

Proposition 7.8 *The counting problem of* 3SAT (i.e., $\#R_{\mathtt{3SAT}}$) *is reducible to computing the permanent of integer matrices. Furthermore, there exists an* even *integer $c > 0$ and a finite set of integers I such that, on input a 3CNF formula ϕ, the reduction produces an integer matrix with entries in I and a permanent value that equals $c^m \cdot \#R_{\mathtt{3SAT}}(\phi)$, where m denotes the number of clauses in ϕ.*

The original proof of Proposition 7.8 uses $c = 2^{10}$ and $I = \{-1, 0, 1, 2, 3\}$. It follows that, for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard (see Exercise 7.3, which also uses Theorem 7.16). Thus, using the case of $c = 2^{10}$, this means that computing the permanent modulo n is NP-hard for any odd $n > 1$. In contrast, computing the permanent modulo 2 (which is equivalent to computing the determinant modulo 2) is easy (i.e., can be done in polynomial-time and even in $\mathcal{NC}$). Thus, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, Proposition 7.8 cannot hold for an odd c (because by Exercise 7.3 it would follow that computing the permanent modulo 2 is NP-Hard). We also note that, assuming $\mathcal{P} \neq \mathcal{NP}$, Proposition 7.8 cannot possibly hold for a set I containing only non-negative integers (see Exercise 7.4).

Proposition 7.9 *Computing the permanent of integer matrices is reducible to computing the permanent of 0/1-matrices. Furthermore, the reduction transforms an integer matrix A into a 0/1-matrix A'' such that the permanent of A can be easily computed from A and the permanent of A''.*

The proofs of Propositions 7.8 and 7.9 are omitted.

7.2 Approximate Counting

Let us consider the counting problem associated with an arbitrary $R \in \mathcal{PC}$. Without loss of generality, we assume that all solutions to n-bit instances have the same length $\ell(n)$, where indeed ℓ is a polynomial. We first note that, while it may be hard to compute $\#R$, given x it is easy to approximate $\#R(x)$ up to $0.01 \cdot 2^{\ell(|x|)}$. Indeed, such an approximation is very rough, but it is not trivial. More generally, we have the following algorithm that produces an estimate of $\#R(x)$ that deviates from the correct value by an additive term that is related to the absolute bound on the number of solutions (i.e., $2^{\ell(|x|)}$).

Proposition 7.10 (approximation with additive deviation): *Let $R \in \mathcal{PC}$ and ℓ be a polynomial such that $R \subseteq \cup_{n\in\mathbb{N}}\{0,1\}^n \times \{0,1\}^{\ell(n)}$. Then, for every polynomial p, there exists a probabilistic polynomial-time algorithm A such that for every $x \in \{0,1\}^*$ and $\delta \in (0,1)$ it holds that*

$$\mathsf{Pr}[|A(x,\delta) - \#R(x)| > (1/p(|x|)) \cdot 2^{\ell(|x|)}] < \delta. \tag{7.1}$$

(As usual, δ is presented to A in binary, and hence the running time of $A(x,\delta)$ is upper-bounded by $\mathrm{poly}(|x| \cdot \log(1/\delta))$.

Proof Sketch: On input x and δ, algorithm A sets $t = \Theta(p(|x|)^2 \cdot \log(1/\delta))$, selects uniformly $y_1, ..., y_t$ and outputs $|\{i : (x, y_i) \in R\}|/t$. □

Discussion. Proposition 7.10 is meaningful in case $\#R(x) > (1/p(|x|)) \cdot 2^{\ell(|x|)}$ holds for some x's. But otherwise, a trivial approximation (i.e., outputting the constant value zero) meets the bound of Eq. (7.1). In general, an approximation of $\#R(x)$ up-to a constant factor (or some other reasonable factor) is more meaningful.[2] In Section 7.2.1, we consider a non-trivial case where such a relative approximation can be obtained in probabilistic polynomial-time. For reasons explained in Section 7.2.1, we do not expect this to happen for every counting problem in $\#\mathcal{P}$, but in Section 7.2.2 we show that relative approximation for any problem in $\#\mathcal{P}$ can be obtained by a randomized Cook-reduction to $\mathcal{NP}$. But before turning to these results, let us state the underlying definition (and actually strengthen it by requiring approximation to within a factor of $1 \pm \varepsilon$).

Definition 7.11 (approximation with relative deviation): *Let $f : \{0,1\}^* \to \mathbb{N}$ and $\varepsilon, \delta : \mathbb{N} \to [0,1]$. A randomized process Π is called an (ε,δ)-*approximator of *f if for every x it holds that*

$$\mathsf{Pr}\left[|\Pi(x) - f(x)| > \varepsilon(|x|) \cdot f(x)\right] \;<\; \delta(|x|). \tag{7.2}$$

We say that f is efficiently $(1-\varepsilon)$-approximable (or just $(1-\varepsilon)$-approximable) *if there exists a probabilistic polynomial-time algorithm A that constitute an $(\varepsilon, 1/3)$-approximator of f.*

The error probability of the latter algorithm A (which has error probability $1/3$) can be reduced to δ by $O(\log(1/\delta))$ repetitions (see Exercise 7.5). Typically, the running time of A will be polynomial in $1/\varepsilon$, and ε is called the deviation parameter.

7.2.1 Relative approximation for $\#R_{\mathtt{dnf}}$

Consider the relation $R_{\mathtt{dnf}}$ consisting of pairs (ϕ,τ) such that ϕ is a DNF formula and τ is an assignment satisfying it. Recall that the search problem of $R_{\mathtt{dnf}}$ is easy to solve and that the proof of Theorem 7.6 establishes that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete (via a non-parsimonious reduction). Still there exists a probabilistic polynomial-time algorithm that provides a constant factor approximation of $\#R_{\mathtt{dnf}}$. We warn that the fact that $\#R_{\mathtt{dnf}}$ is $\#\mathcal{P}$-complete *via a non-parsimonious reduction* means that the constant factor approximation for $\#R_{\mathtt{dnf}}$ does not seem to imply a similar approximation for all problems in $\#\mathcal{P}$. In fact, we should not expect each problem in $\#\mathcal{P}$ to have a (probabilistic) polynomial-time constant-factor approximation algorithm because this would imply $\mathcal{NP} \subseteq \mathcal{BPP}$ (since a constant factor approximation allows for distinguishing the case in which the instance has no solution from the case in which the instance has a solution).

The following algorithm is actually a deterministic reduction of the task of $(\varepsilon,1/3)$-approximating $\#R_{\mathtt{dnf}}$ to the (additive deviation) approximation provided in Proposition 7.10. Consider a DNF formula $\phi = \bigvee_{i=1}^{m} C_i$, where each $C_i : \{0,1\}^n \to \{0,1\}$ is a conjunction. Actually, we will deal with the more general problem in which we are (implicitly) given m subsets $S_1, ..., S_m \subseteq \{0,1\}^n$ and wish to approximate $|\bigcup_i S_i|$. In our case, each S_i is the

[2] We refrain from formally defining an F-factor approximation in this section, although we shall refer to this notion in several informal discussions. There are several ways of defining the aforementioned term (and they are all equivalent when applied to our informal discussions). For example, an F-factor approximation of $\#R$ may mean that, with high probability, the output $A(x)$ satisfies $\#R(x)/F(|x|) \leq A(x) \leq F(|x|) \cdot \#R(x)$. Alternatively, we may require that $\#R(x) \leq A(x) \leq F(|x|) \cdot \#R(x)$ (or, alternatively, that $\#R(x)/F(|x|) \leq A(x) \leq \#R(x)$.

set of assignments satisfying the conjunction C_i. In general, we make two computational assumptions regarding these sets (letting efficient mean implementable in time polynomial in $n \cdot m$):

1. Given $i \in [m]$, one can efficiently determine $|S_i|$.

2. Given $i \in [m]$ and $J \subseteq [m]$, one can efficiently approximate $\mathsf{Pr}_{s \in S_i}\left[s \in \bigcup_{j \in J} S_j\right]$ up to an additive deviation of $1/\mathrm{poly}(n+m)$.

These assumptions are satisfied in our setting (where $S_i = C_i^{-1}(1)$, see Exercise 7.6). The key observation towards approximating $|\bigcup_{i=1}^m S_i|$ is that

$$\left|\bigcup_{i=1}^m S_i\right| \;=\; \sum_{i=1}^m \left|S_i \setminus \bigcup_{j<i} S_j\right| \;=\; \sum_{i=1}^m |S_i| \cdot \mathsf{Pr}_{s \in S_i}\left[s \notin \bigcup_{j<i} S_j\right] \tag{7.3}$$

and that the probabilities in Eq. (7.3) can be approximated by the second assumption. This leads to the following algorithm, where ε denotes the desired deviation parameter (i.e., we wish to obtain $(1 \pm \varepsilon) \cdot |\bigcup_{i=1}^m S_i|$).

Construction 7.12 *Let $\varepsilon' = \varepsilon/m$. For $i = 1$ to m do:*

1. *Using the first assumption, compute $|S_i|$.*

2. *Using the second assumption, obtain $\widetilde{p}_i = (1 \pm \varepsilon') \cdot p_i$, where $p_i \stackrel{\text{def}}{=} \mathsf{Pr}_{s \in S_i}[s \notin \bigcup_{j<i} S_j]$. Set $a_i \stackrel{\text{def}}{=} \widetilde{p}_i \cdot |S_i|$.*

Output the sum of the a_i's.

Let $N_i = p_i \cdot |S_i|$. We are interested in the quality of the approximation to $\sum_i N_i = |\bigcup_i S_i|$ provided by $\sum_i a_i$. Using $a_i = (p_i \pm \varepsilon') \cdot |S_i| = N_i \pm \varepsilon' \cdot |S_i|$ (for all i's), we have $\sum_i a_i = \sum_i N_i \pm \varepsilon' \cdot \sum_i |S_i|$. Using $\sum_i |S_i| \leq m \cdot |\bigcup_i S_i| = m \cdot \sum_i N_i$ (and $\varepsilon = m\varepsilon'$), we get $\sum_i a_i = (1 \pm \varepsilon) \cdot \sum_i N_i$. Thus, we obtain the following result (see Exercise 7.6).

Proposition 7.13 *For every positive polynomial p, the counting problem of $R_{\mathtt{dnf}}$ is efficiently $(1 - (1/p))$-approximable.*

Using the reduction presented in the proof of Theorem 7.6, we conclude that the number of *un*satisfying assignments to a given CNF formula is efficiently $(1-(1/p))$-approximable. We warn, however, that the number of satisfying assignments to such a formula is *not* efficiently approximable. This concurs with the general phenomenon by which *relative approximation may be possible for one quantity, but not for the complementary quantity.* Needless to say, such a phenomenon does not occur in the context of additive-deviation approximation.

7.2.2 Relative approximation for $\#\mathcal{P}$

Recall that we cannot expect to efficiently approximate every $\#\mathcal{P}$ problem. Specifically, efficiently approximating $\#R$ yields an efficient algorithm for deciding membership in $S_R = \{x : R(x) \neq \emptyset\}$. Thus, at best we can hope that approximating $\#R$ is not harder than deciding S_R (i.e., that approximating $\#R$ is reducible in polynomial-time to S_R). This is indeed the case for every NP-complete problem (i.e., if S_R is NP-complete). More generally, we show that approximating any problem in $\#\mathcal{P}$ is reducible in probabilistic polynomial-time to $\mathcal{NP}$.

Theorem 7.14 *For every $R \in \mathcal{PC}$ and positive polynomial p, there exists a probabilistic polynomial-time oracle machine that when given oracle access to $\mathcal{NP}$ constitutes a $(1/p, \mu)$-approximator of $\#R$, where μ is a negligible function* (e.g., $\mu(n) = 2^{-n}$).

Recall that it suffices to provide a $(1/p, \delta)$-approximator of $\#R$, for any constant $\delta < 0.5$, because error reduction is applicable in this context (see Exercise 7.5). Also, it suffices to provide a $(1/2, \delta)$-approximator for every problem in $\#\mathcal{P}$ (see Exercise 7.7).

Proof: Given x, we show how to approximate $|R(x)|$ to within a constant factor. The desired approximation can be obtained as in Exercise 7.7. We may also assume that $R(x) \neq \emptyset$, by starting with the query "is x in S_R" and halting (with output 0) if the answer is negative. Without loss of generality, we assume that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \text{poly}(|x|)$. Our task is to find some $i \in \{1, ..., \ell\}$ such that $2^{i-4} \leq |R(x)| \leq 2^{i+4}$. We proceed in iterations. For $i = 1, ..., \ell + 1$, we find out whether or not $|R(x)| < 2^i$. If the answer is positive then we halt with output 2^i, and otherwise we proceed to the next iteration. (Indeed, if we were able to obtain correct answers to these queries then the output 2^i would satisfy $2^{i-1} \leq |R(x)| < 2^i$.)

Needless to say, the key issue is how to check whether $|R(x)| < 2^i$. The main idea is to use a "random sieve" on the set $R(x)$ such that each element passes the sieve with probability 2^{-i}. Thus, we expect $|R(x)|/2^i$ elements of $R(x)$ to pass the sieve. Assuming that the number of elements in $R(x)$ that pass the random sieve is indeed $\lfloor |R(x)|/2^i \rfloor$, it holds that $|R(x)| \geq 2^i$ if and only if some element of $R(x)$ passes the sieve. Assuming that the sieve can be implemented efficiently, the question of whether or not some element in $R(x)$ passed the sieve is of an "NP-type" (and thus can be referred to our NP-oracle). Combining both assumptions, we may implement the foregoing process by proceeding to the next iteration as long as some element of $R(x)$ passes the sieve. Furthermore, this implementation will provide a reasonably good approximation even if the number of elements in $R(x)$ that pass the random sieve is only approximately equal to $|R(x)|/2^i$. In fact, the level of approximation that this implementation provides is closely related to the level of approximation that is provided by the random sieve. Details follow.

Implementing a random sieve. The random sieve is implemented by using a family of hashing functions (see Appendix). Specifically, in the i^{th} iteration we use a family H_ℓ^i such that each $h \in H_\ell^i$ has a $\text{poly}(\ell)$-bit long description and maps ℓ-bit long strings to i-bit long strings. Furthermore, the family is accompanied with an efficient evaluation algorithm (i.e., mapping adequate pairs (h, x) to $h(x)$) and satisfies (for every $S \subseteq \{0,1\}^\ell$)

$$\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in S : h(y) = 0^i\}| \notin (1 \pm \varepsilon) \cdot 2^{-i}|S|] \;<\; \frac{2^i}{\varepsilon^2 |S|} \tag{7.4}$$

(see Lemma 4.4). The random sieve will let y pass if and only if $h(y) = 0^i$. Indeed, this random sieve is not as perfect as we assumed in the foregoing discussion, but Eq. (7.4) says that in some sense this sieve is good enough.

Implementing the queries. Recall that for some x, i and $h \in H_\ell^i$, we need to determine whether $\{y \in R(x) : h(y) = 0^i\} = \emptyset$. This type of question can be cast as membership in the set

$$S_{R,H} \stackrel{\text{def}}{=} \{(x, i, h) : \exists y \text{ s.t. } (x, y) \in R \wedge h(y) = 0^i\}. \tag{7.5}$$

Using the hypotheses that $R \in \mathcal{PC}$ and that the family of hashing functions has an efficient evaluation algorithm, it follows that $S_{R,H}$ is in $\mathcal{NP}$.

The actual procedure. On input $x \in S_R$ and oracle access to $S_{R,H}$, we proceed in iterations, starting with $i = 1$ and halting at $i = \ell$ (if not before), where ℓ denotes the length of the potential solutions for x. In the i^{th} iteration (where $i < \ell$), we uniformly select $h \in H_\ell^i$ and query the oracle on whether or not $(x, i, h) \in S_{R,H}$. If the answer is negative then we halt with output 2^i, and otherwise we proceed to the next iteration (using $i \leftarrow i + 1$). Needless to say, if we reach the last iteration (i.e., $i = \ell$) then we just halt with output 2^ℓ.

Indeed, we have ignored the case that $x \notin S_R$, which can be easily handled by a minor modification of the foregoing procedure. Specifically, on input x, we first query S_R on x and halt with output 0 if the answer is negative. Otherwise we proceed as in the foregoing procedure.

The analysis. We upper-bound separately the probability that the procedure outputs a value that is too small and the probability that it outputs a value that is too big. In light of the foregoing discussion, we may assume that $|R(x)| > 0$, and let $i_x = \lfloor \log_2 |R(x)| \rfloor \geq 0$.

1. The probability that the procedure *halts in a specific iteration* $i < i_x$ equals $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| = 0]$, which in turn is upper-bounded by $2^i/|R(x)|$ (using Eq. (7.4) with $\varepsilon = 1$). Thus, the probability that the procedure halts *before* iteration $i_x - 3$ is upper-bounded by $\sum_{i=0}^{i_x-4} 2^i/|R(x)|$, which in turn is less than $1/8$ (because $i_x \leq \log_2 |R(x)|$). Thus, with probability at least $7/8$, the output is at least $2^{i_x-3} > |R(x)|/16$ (because $i_x > (\log_2 |R(x)|) - 1$).

2. The probability that the procedure *does not halt in iteration* $i > i_x$ equals $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$, which in turn is upper-bounded by $\alpha/(\alpha - 1)^2$, where $\alpha = 2^i/|R(x)| > 1$ (using Eq. (7.4) with $\varepsilon = \alpha - 1$).[3] Thus, the probability that the procedure does not halt by iteration $i_x + 4$ is upper-bounded by $8/49 < 1/6$ (because $i_x > (\log_2 |R(x)|) - 1$). Thus, with probability at least $5/6$, the output is at most $2^{i_x+4} \leq 16 \cdot |R(x)|$ (because $i_x \leq \log_2 |R(x)|$).

Thus, with probability at least $(7/8) - (1/6) > 2/3$, the foregoing procedure outputs a value v such that $v/16 \leq |R(x)| < 16v$. Reducing the deviation by using the ideas presented in Exercise 7.7 (and reducing the error probability as in Exercise 7.5), the theorem follows. ■

[3] A better bound can be obtained by using the hypothesis that, for every y, when h is uniformly selected in H_ℓ^i, the value of $h(y)$ is uniformly distributed in $\{0,1\}^i$. In this case, $\mathsf{Pr}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}| \geq 1]$ is upper-bounded by $\mathsf{E}_{h \in H_\ell^i}[|\{y \in R(x) : h(y) = 0^i\}|] = |R(x)|/2^i$.

Perspective. The key observation underlying the proof Theorem 7.14 is that while we cannot test (even with the help of an NP-oracle) whether the number of solutions is greater than a given number, we can test (with the help of an NP-oracle) whether the number of solutions that "survive a random sieve" is greater than zero. If fact, we can also test whether the number of solutions that "survive a random sieve" is greater than a small number, where small means polynomial in the length of the input (see Exercise 7.9). In general, our complexity is linear in the size of the threshold, and not in the length of its binary description. Indeed, in many settings it is more advantageous to use a threshold that is polynomial in some efficiency parameter (rather than using the threshold zero); examples appear in Section 7.4.2 and in [49].

7.3 Searching for unique solutions

A natural computational problem (regarding search problems), which arises when discussing the number of solutions, is the problem of distinguishing instances having a single solution from instances having no solution (or finding the unique solution whenever such exists). We mention that instances having a single solution facilitate numerous arguments (see, for example, Exercise 7.3 and [21]). Formally, searching for and deciding the existence of unique solutions are defined within the framework of promise problems.

Definition 7.15 (search and decision problems for unique solution instances): *The* set of instances having unique solutions *with respect to the binary relation R is defined as* $\mathtt{US}_R \stackrel{\text{def}}{=} \{x : |R(x)| = 1\}$, *where* $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$. *As usual, we denote* $S_R = \{x : |R(x)| \geq 1\}$, *and* $\overline{S}_R \stackrel{\text{def}}{=} \{0,1\}^* \setminus S_R = \{x : |R(x)| = 0\}$.

- *The problem of* finding unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R \cup \overline{S}_R$.[4]

  In continuation to the notion of candid search problems, the candid searching for unique solutions for R *is defined as the* search problem R with promise $\mathtt{US}_R$.

[4] A search problem with a promise consists of a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and a promise set P. Such a problem is also referred to as the search problem R with promise P.

- The search problem R with promise P is solved by algorithm A if for every $x \in P$ it holds that $(x, A(x)) \in R$ if $x \in S_R = \{x : R(x) \neq \emptyset\}$ and $A(x) = \perp$ otherwise, where $R(x) = \{y : (x,y) \in R\}$.

  The time complexity of A on inputs in P is defined as $T_{A|P}(n) \stackrel{\text{def}}{=} \max_{x \in P \cap \{0,1\}^n}\{t_A(x)\}$, where $t_A(x)$ is the running time of $A(x)$ and $T_{A|P}(n) = 0$ if $P \cap \{0,1\}^n = \emptyset$.
- The search problem R with promise P is in the promise problem extension of $\mathcal{PF}$ if there exists a polynomial-time algorithm that solves this problem.
- The search problem R with promise P is in the promise problem extension of $\mathcal{PC}$ if there exists a polynomial T and an algorithm A such that, for every $x \in P$ and $y \in \{0,1\}^*$, algorithm A makes at most $T(|x|)$ steps and it holds that $A(x,y) = 1$ if and only if $(x,y) \in R$.

An algorithm A solves the candid search problem of the binary relation R if for every $x \in S_R \stackrel{\text{def}}{=} \{x : \exists y \text{ s.t. } (x,y) \in R\}$ it holds that $(x, A(x)) \in R$. The time complexity of such an algorithm is defined as $T_{A|S_R}(n) \stackrel{\text{def}}{=} \max_{x \in P \cap \{0,1\}^n}\{t_A(x)\}$, where $t_A(x)$ is the running time of $A(x)$ and $T_{A|S_R}(n) = 0$ if $P \cap \{0,1\}^n = \emptyset$.

- *The problem of* deciding unique solution for R *is defined as the* promise problem $(\mathtt{US}_R, \overline{S}_R)$.

Interestingly, in many natural cases, the promise does not make any of these problems any easier than the original problem. That is, for all known NP-complete problems, the original problem is reducible in probabilistic polynomial-time to the corresponding unique instances problem.

Theorem 7.16 *Let $R \in \mathcal{PC}$ and suppose that every search problem in $\mathcal{PC}$ is parsimoniously reducible to R. Then solving the search problem of R* (resp., deciding membership in S_R) *is reducible in probabilistic polynomial-time to finding unique solutions for R* (resp., to the promise problem $(\mathtt{US}_R, \overline{S}_R)$). *Furthermore, there exists a probabilistic polynomial-time computable mapping M such that for every $x \in \overline{S}_R$ it holds that $M(x) \in \overline{S}_R$, whereas for every $x \in S_R$ it holds that $\mathsf{Pr}[M(x) \in \mathtt{US}_R] \geq 1/\mathrm{poly}(|x|)$.*

We note that the condition regarding parsimonious reductions is crucial (see Exercise 7.10).

Proof: As in the proof of Theorem 7.14, the idea is to apply a "random sieve" on $R(x)$, this time with the hope that a single element survives. Specifically, if we let each element passes the sieve with probability approximately $1/|R(x)|$ then with constant probability a single element survives. Sieving will be performed by a random function selected in an adequate hashing family (see Appendix). A couple of questions arise:

1. *How do we get an approximation to $|R(x)|$?* Note that we need such an approximation in order to determine the adequate hashing family. Indeed, we may just invoke Theorem 7.14, but this will not yield a many-to-one reduction. Instead, we just select $m \in \{0, ..., \mathrm{poly}(|x|)\}$ uniformly and note that (if $|R(x)| > 0$ then) $\mathsf{Pr}[m = \lceil \log_2 |R(x)| \rceil] = 1/\mathrm{poly}(|x|)$.

   Thus, we randomly map x to (x, m, h), where h is uniformly selected in an adequate hashing family.

2. *How does the question of whether a single element of $R(x)$ pass the random sieve translate to an instance of the unique-instance problem for R?* Recall that in the proof of Theorem 7.14 the non-emptiness of the set of element of $R(x)$ that pass the sieve defined by h was determined by checking membership (of (x, m, h)) in $S_{R,H} \in \mathcal{NP}$ (defined in Eq. (7.5)). Furthermore, the number of NP-witnesses for $(x, m, h) \in S_{R,H}$ equals the number of elements of $R(x)$ that pass the sieve. Using the parsimonious reduction of $S_{R,H}$ to S_R (which is guaranteed by the theorem's hypothesis), we obtained the desired instance.

Note that in case $R(x) = \emptyset$ the aforementioned mapping always generates a no-instance (of $S_{R,H}$ and thus of S_R). Details follow.

Implementation (i.e., the mapping M). As in the proof of Theorem 7.14, we assume, without loss of generality, that $R(x) \subseteq \{0,1\}^\ell$, where $\ell = \mathrm{poly}(|x|)$. We start by uniformly selecting $m \in \{0, 1, ..., \ell\}$ and $h \in H_\ell^m$, where H_ℓ^m is a family of efficiently computable and pairwise-independent hashing functions (see Definition 4.1) mapping ℓ-bit long strings to m-bit long

strings.[5] Thus, we obtain an instance (x,m,h) of $S_{R,H} \in \mathcal{NP}$ such that the set of valid solutions for (x,m,h) equals $\{y \in R(x) : h(y) = 0^m\}$. Using the parsimonious reduction g of $S_{R,H}$ to S_R, we map (x,m,h) to $g(x,m,h)$, and it holds that $|\{y \in R(x) : h(y) = 0^m\}|$ equals $|R(g(x,m,h))|$. To summarize, on input x the randomized mapping M outputs the instance $M(x) \stackrel{\text{def}}{=} g(x,m,h)$, where $m \in \{0,1,...,\ell\}$ and $h \in H_\ell^m$ are uniformly selected.

The analysis. Note that for any $x \in \overline{S}_R$ it holds that $\mathsf{Pr}[M(x) \in \overline{S}_R] = 1$. Assuming that $x \in S_R$, with probability exactly $1/(\ell+1)$ it holds that $m = m_x$, where $m_x \stackrel{\text{def}}{=} \lceil \log_2 |R(x)| \rceil$. In this case, for a uniformly selected $h \in H_\ell^m$, we lower-bound the probability that $\{y \in R(x) : h(y) = 0^m\}$ is a singleton. Using the Inclusion-Exclusion Principle, we have

$$\begin{aligned} &\mathsf{Pr}_{h \in H_\ell^{m_x}}[|\{y \in R(x) : h(y) = 0^{m_x}\}| = 1] \qquad (7.6) \\ &\geq \sum_{y \in R(x)} \mathsf{Pr}_{h \in H_\ell^{m_x}}[h(y) = 0^{m_x}] \\ &\quad - \sum_{y_1 < y_2 \in R(x)} \mathsf{Pr}_{h \in H_\ell^{m_x}}[h(y_1) = h(y_2) = 0^{m_x}] \\ &= |R(x)| \cdot 2^{-m_x} - \binom{|R(x)|}{2} \cdot 2^{-2m_x} \end{aligned}$$

where the equality is due to the pairwise independence property. Using $2^{m_x - 1} < |R(x)| \leq 2^{m_x}$, it follows that Eq. (7.6) is lower-bounded by $1/4$. Thus, $\mathsf{Pr}[M(x) \in \mathtt{US}_R] \geq 1/4(\ell+1)$, and the theorem follows. ∎

Comment. Theorem 7.16 is sometimes stated as referring to the unique solution problem of SAT. In this case and when using a specific family of pairwise independent hashing functions, the use of the parsimonious reduction can be avoided. For details see Exercise 7.11.

7.4 Uniform generation of solutions

We now turn to a new type of computational problems, which may be viewed as a straining of search problems. We refer to the task of generating a uniformly distributed solution for a given instance, rather than merely finding an adequate solution. Needless to say, by definition, algorithms solving this ("uniform generation") task must be randomized. Focusing on relations in $\mathcal{PC}$ we consider two versions of the problem, which differ by the level of approximation provided for the desired (uniform) distribution.[6]

Definition 7.17 (uniform generation): *Let $R \in \mathcal{PC}$ and $S_R = \{x : |R(x)| \geq 1\}$, and let Π be a probabilistic process.*

[5] For sake of uniformity, we allow also the case of $m = 0$, which is rather artificial. In this case all hashing functions in H_ℓ^0 map $\{0,1\}^\ell$ to the empty string, which is viewed as 0^0.

[6] Note that a probabilistic algorithm running in strict polynomial-time is not able to output a perfectly uniform distribution on sets of certain sizes. Specifically, referring to the standard model that allows only for uniformly selected binary values, such algorithms cannot output a perfectly uniform distribution on sets having cardinality that is not a power of two.

1. *We say that* Π solves the uniform generation problem of R *if, on input* $x \in S_R$*, the process* Π *outputs either an element of* $R(x)$ *or a special symbol, denoted* $\perp$*, such that* $\mathsf{Pr}[\Pi(x) \in R(x)] \geq 1/2$ *and for every* $y \in R(x)$ *it holds that* $\mathsf{Pr}[\Pi(x) = y \mid \Pi(x) \in R(x)] = 1/|R(x)|$.

2. *For* $\varepsilon : \mathbb{N} \to [0,1]$*, we say that* Π solves the $(1-\varepsilon)$-approximate uniform generation problem of R *if, on input* $x \in S_R$*, the distribution* $\Pi(x)$ *is* $\varepsilon(|x|)$*-close to the uniform distribution on* $R(x)$.

In both cases, without loss of generality, we may require that if $x \notin S_R$ *then* $\mathsf{Pr}[\Pi(x) = \perp] = 1$. *More generally, we may require that* Π *never outputs a string not in* $R(x)$.

Note that the error probability of uniform generation (as in Item 1) can be made exponentially vanishing (in $|x|$) by employing error-reduction. In contrast, we are not aware of any general way of reducing the deviation of an approximate uniform generation procedure (as in Item 2).[7]

In Section 7.4.1 we show that, for many search problems, approximate uniform generation is computationally equivalent to approximate counting. In Section 7.4.2 we present a direct approach for solving the uniform generation problem of any search problem in $\mathcal{PC}$ by using an oracle to $\mathcal{NP}$.

7.4.1 Relation to approximate counting

We show that for every $R \in \mathcal{PC}$ that is NP-complete under parsimonious reductions, the approximate counting problem associated with R is computationally equivalent to approximate uniform generation with respect to R. Recalling that both approximate problems are parameterized by the level of precision, we obtain the following quantitative form of the aforementioned equivalence.

Theorem 7.18 *Let* $R \in \mathcal{PC}$ *and let* ℓ *be a polynomial such that for every* $(x,y) \in R$ *it holds that* $|y| \leq \ell(|x|)$*. Suppose that every search problem in* $\mathcal{PC}$ *is parsimoniously reducible to* R.

1. From approximate counting to approximate uniform generation: *Let* $\varepsilon(n) = 1/5\ell(n)$ *and let* $\mu : \mathbb{N} \to (0,1)$ *be a function satisfying* $\mu(n) \geq \exp(-\mathrm{poly}(n))$*. Then,* $(1-\mu)$*-approximate uniform generation for* R *is reducible in probabilistic polynomial-time to* $(1-\varepsilon)$*-approximating* $\#R$.

2. From approximate uniform generation to approximate counting: *For every noticeable* $\varepsilon : \mathbb{N} \to (0,1)$ (i.e., $\varepsilon(n) \geq 1/\mathrm{poly}(n)$ for every n)*, the problem of* $(1-\varepsilon)$*-approximating* $\#R$ *is reducible in probabilistic polynomial-time to* $(1-\varepsilon')$*-approximate uniform generation problem of* R*, where* $\varepsilon'(n) = \varepsilon(n)/5\ell(n)$.

Note that the quality of the approximate uniform generation asserted in Part 1 (i.e., μ) is independent of the quality of the approximate counting procedure (i.e., ε) to which the former is reduced, provided that the approximate counter performs better than some threshold. On the other hand, the quality of the approximate counting asserted in Part 2 (i.e., ε) does

[7] We note that in some cases, the deviation of an approximate uniform generation procedure can be reduced. See discussion following Theorem 7.18.

depend on the quality of the approximate uniform generation (i.e., ε'). Recall, however, that the quality of approximate counting procedures for problems that are NP-complete under parsimonious reductions can be improved (see Exercise 7.8). Thus, for such problems, the quality of approximate uniform generation procedures can be improved by applying both parts of Theorem 7.18.

Proof: Throughout the proof, we assume for simplicity (and in fact without loss of generality) that $R(x) \neq \emptyset$ and $R(x) \subseteq \{0,1\}^{\ell(|x|)}$.

Towards Part 1, let use first reduce the uniform generation problem of R to $\#R$ (rather than to approximating $\#R$). On input $x \in S_R$, we generate a uniformly distributed $y \in R(x)$ by randomly generating its bits one after the other. We proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x;y') \stackrel{\text{def}}{=} \{y'' : (x, y'y'') \in R\}$ is not empty. With probability $|R'(x;y'1)|/|R'(x;y')|$ we set the i^{th} bit to equal 1, and otherwise we set it to equal 0. We obtain both $|R'(x;y'1)|$ and $|R'(x;y')|$ by using a parsimonious reduction g of $R' = \{((x;y'),y'') : (x,y'y'') \in R\} \in \mathcal{PC}$ to R. That is, we obtain $|R'(x;y')|$ by querying for the value of $|R(g(x;y'))|$. Ignoring integrality issues, all this works perfectly (i.e., we generate an $\ell(n)$-bit string uniformly distributed in $R(x)$) as long as we have oracle access to $\#R$. But we only have oracle access to an approximation of $\#R$, and thus a careful modification is in place.

Let us denote the approximation oracle by A. Firstly, by adequate error reduction, we may assume that, for every x, it holds that $\mathsf{Pr}[A(x) \in (1 \pm \varepsilon(n)) \cdot \#R(x)] > 1 - \mu'(|x|)$, where $\mu'(n) = \mu(n)/\ell(n)$. In the rest of the analysis we ignore the probability that the estimate deviates from the aforementioned interval, and note that this rare event is the only source of the possible deviation of the output distribution from the uniform distribution on $R(x)$.[8] Let us assume for a moment that A is *deterministic* and that for every x and y' it holds that

$$A(g(x,y'0)) + A(g(x,y'1)) \;\leq\; A(g(x;y')). \tag{7.7}$$

We also assume that the approximation is correct at the "trivial level" (where one may just check whether or not (x,y) is in R); that is, for every $y \in \{0,1\}^{\ell(|x|)}$, it holds that

$$A(g(x;y)) = 1 \text{ if } (x,y) \in R \text{ and } A(g(x;y)) = 0 \text{ otherwise.} \tag{7.8}$$

We modify the i^{th} iteration of the foregoing procedure such that, when entering with the $(i-1)$-bit long prefix y', we set the i^{th} bit to 1 (resp., to 0) with probability $A(g(x;y'1))/A(g(x;y'))$ (resp., with probability $A(g(x;y'0))/A(g(x;y'))$) and halt (with output $\perp$) with the residual probability. If we completed the last (i.e., $\ell(|x|)^{\text{th}}$) iteration, then we output the $\ell(|x|)$-bit long string that was generated. Thus, as long as Eq. (7.7) holds (but regardless of other aspects of the quality of the approximation), every $y = \sigma_1 \cdots \sigma_{\ell(|x|)} \in R(x)$, is output with probability

$$\frac{A(g(x;\sigma_1))}{A(g(x;\lambda))} \cdot \frac{A(g(x;\sigma_1\sigma_2))}{A(g(x;\sigma_1))} \cdots \frac{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)}))}{A(g(x;\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)-1}))} \tag{7.9}$$

which, by Eq. (7.8), equals $1/A(g(x;\lambda))$. Thus, the procedure outputs each element of $R(x)$ with equal probability, and never outputs a non-$\perp$ value that is outside $R(x)$. It follows

[8] The possible deviation is due to the fact that this rare event may occur with different probability in the different invocations of algorithm A.

that the quality of approximation only effects the probability that the procedure outputs a non-$\perp$ value (which equals $|R(x)|/A(g(x;\lambda))$).

We now turn to enforcing Eq. (7.7) and Eq. (7.8). We may enforce Eq. (7.8) by performing the straightforward check (of whether or not $(x,y) \in R$) rather than invoking $A(g(x,y))$.[9] As for Eq. (7.7), we enforce it artificially by using $A'(x,y') \stackrel{\text{def}}{=} (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot A(g(x;y'))$ instead of $A(g(x;y'))$. Recalling that $A(g(x;y')) = (1 \pm \varepsilon(|xy'|)) \cdot |R'(x;y')|$, we have

$$\begin{aligned} A'(x,y') &> (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|)} \cdot (1-\varepsilon(|x|)) \cdot |R'(x;y')| \\ A'(x,y'\sigma) &< (1+\varepsilon(|x|))^{3(\ell(|x|)-|y'|-1)} \cdot (1+\varepsilon(|x|)) \cdot |R'(x;y'\sigma)| \end{aligned}$$

and the claim follows using $(1-\varepsilon(|x|)) \cdot (1+\varepsilon(|x|))^3 > (1-\varepsilon(|x|))$. Note that the foregoing modification only decreases the probability of outputting a non-$\perp$ value by a factor of $(1+\varepsilon(|x|))^{3\ell(|x|)} < 2$, where the inequality is due to the setting of ε (i.e., $\varepsilon(n) = 1/5\ell(n)$). Finally, we refer to our assumption that A is deterministic. This assumption was only used in order to identify the value of $A(g(x,y'))$ obtained and used in the $(|y'|-1)^{\text{st}}$ iteration with the value of $A(g(x,y'))$ obtained and used in the $|y'|^{\text{th}}$ iteration, but the same effect can be obtained by just using the former value (in the $|y'|^{\text{th}}$ iteration) rather than re-invoking A in order to obtain it. Part 1 follows.

Towards Part 2, let use first reduce the task of approximating $\#R$ to the task of (exact) uniform generation for R. On input $x \in S_R$, the reduction uses the tree of possible prefixes of elements of $R(x)$ in a somewhat different manner. Again, we proceed in iterations, entering the i^{th} iteration with an $(i-1)$-bit long string y' such that $R'(x;y') \stackrel{\text{def}}{=} \{y'' : (x,y'y'') \in R\}$ is not empty. At the i^{th} iteration we estimate the bigger among the two fractions $|R'(x;y'0)|/|R'(x;y')|$ and $|R'(x;y'1)|/|R'(x;y')|$, by uniformly sampling the uniform distribution over $R'(x;y')$. That is, taking $\text{poly}(|x|/\varepsilon'(|x|))$ uniformly distributed samples in $R'(x;y')$, we obtain with overwhelmingly high probability an approximation of these fractions up to an additive deviation of at most $\varepsilon'(|x|)/3$. This means that we obtain a relative approximation up-to a factor of $1 \pm \varepsilon'(|x|)$ for the fraction (or fractions) that is (resp., are) bigger than $1/3$. Indeed, we may not be able to obtain such a good relative approximation of the other fraction (in case it is very small), but this does not matter. It also does not matter that we cannot tell which is the bigger fraction among the two; it only matter that we use an approximation that indicates a quantity that is, say, bigger than $1/3$. We proceed to the next iteration by augmenting y' using the bit that corresponds to such a quantity. Specifically, suppose that we obtained the approximations $a_0(y') \approx |R'(x;y'0)|/|R'(x;y')|$ and $a_1(y') \approx |R'(x;y'1)|/|R'(x;y')|$. Then we extend y' by the bit 1 if $a_1(y') > a_0(y')$ and extend y' by the bit 0 otherwise. Finally, when we reach $y = \sigma_1 \cdots \sigma_{\ell(|x|)}$ such that $(x,y) \in R$, we output

$$a_{\sigma_1}(\lambda)^{-1} \cdot a_{\sigma_2}(\sigma_1)^{-1} \cdots a_{\sigma_{\ell(|x|)}}(\sigma_1\sigma_2\cdots\sigma_{\ell(|x|)-1})^{-1}. \tag{7.10}$$

As in Part 1, actions regarding R' (in this case uniform generation in R') are conducted via the parsimonious reduction g to R. That is, whenever we need to sample uniformly

[9] Alternatively, we note that since A is a $(1-\varepsilon)$-approximator for $\varepsilon < 1$ it must hold that $\#R'(z) = 0$ implies $A(z) = 0$. Also, since $\varepsilon < 1/3$, if $\#R'(z) = 1$ then $A(z) \in (2/3, 4/3)$, which may be rounded to 1.

in the set $R'(x;y')$, we sample the set $R(g(x;y'))$ instead. Finally, note that the deviation from uniform distribution (i.e., the fact that we can only approximately sample R) merely introduces such a deviation in each of our approximations to the relevant fractions (i.e., to a fraction bigger than 1/3). Specifically, on input x, using an oracle that provides a $(1-\varepsilon')$-approximate uniform generation for R, with overwhelmingly high probability, the output (as defined in Eq. (7.10)) is in

$$\prod_{i=1}^{\ell(|x|)} \left((1 \pm 2\varepsilon'(|x|)) \cdot \frac{|R'(x;\sigma_1 \cdots \sigma_{i-1})|}{|R'(x;\sigma_1 \cdots \sigma_i)|} \right) \tag{7.11}$$

where the error probability is due to the unlikely case that in one of the iterations our approximations deviates from the correct value by more than an additive deviation term of $\varepsilon'(n)/3$. Noting that Eq. (7.11) equals $(1\pm2\varepsilon'(|x|))^{\ell(|x|)} \cdot |R(x)|$ and using $(1\pm2\varepsilon'(|x|))^{\ell(|x|)} \subset (1 \pm \varepsilon(|x|))$, Part 2 follows, and so does the theorem. ■

7.4.2 Direct uniform generation

We conclude the current section by presenting a direct procedure for solving the uniform generation problem of any $R \in \mathcal{PC}$. This procedure uses an oracle to $\mathcal{NP}$, which is unavoidable because solving the uniform generation problem implies solving the corresponding search problem. One advantage of this process, over the reduction presented in Section 7.4.1, is that it solves the uniform generation problem rather than the *approximate* uniform generation problem.

We are going to use hashing again, but this time we use a family of hashing functions having a stronger "uniformity property" (see Section 4.3). Specifically, we will use a family of ℓ-wise independent hashing functions mapping ℓ-bit strings to m-bit strings, where ℓ bounds the length of solutions in R, and rely on the fact that such a family satisfies Lemma 4.6. Intuitively, such functions partition $\{0,1\}^\ell$ into 2^m cells and Lemma 4.6 asserts that these partitions "uniformly shatter" all sufficiently large sets. That is, for every set $S \subseteq \{0,1\}^\ell$ of size $\Omega(\ell \cdot 2^m)$ the partition induced by almost every function is such that each cell contains approximately $|S|/2^m$ elements of S. In particular, if $|S| = \Theta(\ell \cdot 2^m)$ then each cell contains $\Theta(\ell)$ elements of S.

In the following construction, we assume that on input x we also obtain a good approximation to the size of $R(x)$. This assumption can be enforced by using an approximate counting procedure as a preprocessing stage. Alternatively, the ideas presented in the following construction yield such an approximate counting procedure.

Construction 7.19 (uniform generation): *On input x and $m'_x \in \{m_x, m_x+1\}$, where $m_x \stackrel{\text{def}}{=} \lfloor \log_2 |R(x)| \rfloor$ and $R(x) \subseteq \{0,1\}^\ell$, the oracle machine proceeds as follows.*

1. Selecting a partition that "uniformly shatters" $R(x)$. *The machine sets $m = \max(0, m'_x - 6 - \log_2 \ell)$ and selects uniformly $h \in H^m_\ell$. Such a function defines a partition of $\{0,1\}^\ell$ into 2^m cells[10], and the hope is that each cell contains approximately the same elements of $R(x)$. Next, the machine checks that this is indeed the case or rather than*

[10] For sake of uniformity, we allow also the case of $m = 0$, which is rather artificial. In this case all hashing functions in H^0_ℓ map $\{0,1\}^\ell$ to the empty string, which is viewed as 0^0, and thus define a trivial partition of $\{0,1\}^\ell$ (i.e., into a single cell).

no cell contains more that 1000ℓ *elements of* $R(x)$*. This is done by checking whether or not* $(x,h,1^{1000\ell})$ *is in the set* $S^{(1)}_{R,H}$ *defined as follows*

$$\begin{aligned} S^{(1)}_{R,H} &\stackrel{\text{def}}{=} \{(x',h',1^t) : \exists v \text{ s.t. } |\{y : (x',y)\in R \wedge h'(y)=v\}| \geq t\} \\ &= \{(x',h',1^t) : \exists v,y_1,...,y_t \text{ s.t. } \psi^{(1)}(x',h',v,y_1,...,y_t)\}, \end{aligned} \tag{7.12}$$

where $\psi^{(1)}(x',h',v,y_1,...,y_t)$ *holds if and only if* $y_1<y_2\cdots<y_t$ *and for every* $j\in[t]$ *it holds that* $(x',y_j)\in R\wedge h'(y_j)=v$*. Note that* $S^{(1)}_{R,H}\in\mathcal{NP}$*.*

If the answer is positive (i.e., there exists a cell that contains more that 1000ℓ elements of $R(x)$) *then the machine halts with output* $\perp$*. Otherwise, the machine continues with this choice of* h*. In this case, for every* $v\in\{0,1\}^m$*, it holds that no cell contains more that* 1000ℓ *elements of* $R(x)$ (i.e., $|\{y:(x,y)\in R\wedge h(y)=v\}|<1000\ell$)*. We stress that this is an absolute guarantee that follows from* $(x,h,1^{1000\ell})\notin S^{(1)}_{R,H}$*.*

2. Selecting a cell and determining the number of elements of $R(x)$ that are contained in it. *The machine selects uniformly* $v\in\{0,1\}^m$ *and determines* $s_v\stackrel{\text{def}}{=}|\{y:(x,y)\in R\wedge h(y)=v\}|$ *by making queries to the following NP-set*

$$S^{(2)}_{R,H}\stackrel{\text{def}}{=}\{(x',h',v',1^t):\exists y_1,...,y_t \text{ s.t. } \psi^{(1)}(x',h',v',y_1,...,y_t)\}. \tag{7.13}$$

Specifically, for $i=1,...,1000\ell$*, it checks whether* $(x,h,v,1^i)$ *is in* $S^{(2)}_{R,H}$*, and sets* s_v *to be the largest value of* i *for which the answer is positive.*

3. Obtaining all the elements of $R(x)$ that are contained in the selected cell, and outputting one of them at random. *Using* s_v*, the procedure reconstructs the set* $S_v\stackrel{\text{def}}{=}\{y:(x,y)\in R\wedge h(y)=v\}$*, by making queries to the following NP-set*

$$S^{(3)}_{R,H}\stackrel{\text{def}}{=}\{(x',h',v',1^t,j):\exists y_1,...,y_t \text{ s.t. } \psi^{(3)}(x',h',v',y_1,...,y_t,j)\}, \tag{7.14}$$

where $\psi^{(3)}(x',h',v',y_1,...,y_t,j)$ *holds if and only if* $\psi^{(1)}(x',h',v',y_1,...,y_t)$ *holds and the* j^{th} *bit of* $y_1\cdots y_t$ *equals* 1*. Specifically, for* $j_1=1,...,s_v$ *and* $j_2=1,...,\ell$*, we make the query* $(x,h,v,1^{s_v},(j_1-1)\cdot\ell+j_2)$ *in order to determine the* j_2^{th} *bit of* y_{j_1}*. Finally, having recovered* S_v*, the procedure outputs each* $y\in S_v$ *with probability* $1/1000\ell$*, and outputs* $\perp$ *otherwise.*

By Lemma 4.6 (and $m\leq m_x+1-6-\log_2\ell$), with overwhelmingly high probability, each set $\{y:(x,y)\in R\wedge h(y)=v\}$ has cardinality $(1\pm0.5)|R(x)|/2^m$. Using $m'_x>(\log_2|R(x)|)-1$ (resp., $m'_x\leq(\log_2|R(x)|)+1$), it follows that $|R(x)|/2^m<1000\ell$ (resp., $|R(x)|/2^m>10\ell$). Thus, Step 1 can be easily adapted to yield an approximate counting procedure for $\#R$ (see Exercise 7.12). However, our aim is to establish the following fact.

Proposition 7.20 *Construction 7.19 solves the uniform generation problem of* R*.*

Proof: By Lemma 4.6 (and the setting of m), with overwhelmingly high probability, a uniformly selected $h\in H^m_\ell$ partitions $R(x)$ into 2^m cells, each containing at most 1000ℓ

elements. The key observation, stated in Step 1, is that if the procedure does not halt in Step 1 then it is indeed the case that h induces such a partition. The fact that these cells may contain a different number of elements is immaterial, because each element is output with the same probability (i.e., $1/1000\ell$). What matters is that the average number of elements in the cells is sufficiently large, because this average number determines the probability that the procedure outputs an element of $R(x)$ (rather than $\perp$). Specifically, the latter probability equals the aforementioned average number (which equals $|R(x)|/2^m$) divided by 1000ℓ. Using $m \leq \max(0, \log_2(2|R(x)|) - 6 - \log_2 \ell)$, we have $|R(x)|/2^m \geq \max(1, 32\ell)$, which means that the procedure outputs some element of $R(x)$ with probability at least $1/1000\ell$. ■

Technical comments. We can easily improve the performance of Construction 7.19 by dealing separately with the case $m = 0$. In such a case, Step 3 can be simplified and improved by uniformly selecting and outputting an element of S_λ (which equals $R(x)$). Under this modification, the procedure outputs some element of $R(x)$ with probability at least $1/6$. In any case, recall that the probability that a uniform generation procedure outputs $\perp$ can be deceased by repeated invocations.

Digest. Construction 7.19 is the culmination of the "hashing paradigm" that is aimed at allowing various manipulations of arbitrary sets. In particular, as seen in Construction 7.19, hashing can be used in order to partition a large set into an adequate number of small subsets that are of approximately the same size. We stress that hashing is performed by randomly selecting a function in an adequate family. Indeed, the use of randomization for such purposes (i.e., allowing manipulation of large sets) seems indispensable.

Notes

The counting class $\#\mathcal{P}$ was introduced by Valiant [106], who proved that computing the permanent of 0/1-matrices is $\#\mathcal{P}$-complete (i.e., Theorem 7.7).

The approximation procedure for $\#\mathcal{P}$ is due to Stockmeyer [99], following an idea of Sipser [96]. Our exposition, however, follows further developments in the area. The randomized reduction of $\mathcal{NP}$ to problems of unique solutions was discovered by Valiant and Vazirani [110]. Again, our exposition is a bit different.

The connection between approximate counting and uniform generation (presented in Section 7.4.1) was discovered by Jerrum, Valiant, and Vazirani [64], and turned out to be very useful in the design of algorithms (e.g., in the "Markov Chain approach" (see [83, Sec. 11.3.1])). The direct procedure for uniform generation (presented in Section 7.4.2) is taken from [18].

In continuation to Section 7.2.1, which is based on [67], we refer the interested reader to [63], which presents a probabilistic polynomial-time algorithm for approximating the permanent of non-negative matrices. This fascinating algorithm is based on the fact that knowing (approximately) certain parameters of a non-negative matrix M allows to approximate the same parameters for a matrix M', provided that M and M' are sufficiently similar.

Specifically, M and M' may differ only on a single entry, and the ratio of the corresponding values must be sufficiently close to one. Needless to say, the actual observation (is not generic but rather) refers to specific parameters of the matrix, which include its permanent. Thus, given a matrix M for which we need to approximate the permanent, we consider a sequence of matrices $M_0, ..., M_t \approx M$ such that M_0 is the all 1's matrix (for which it is easy to evaluate the said parameters), and each M_{i+1} is obtained from M_i by reducing some adequate entry by a factor sufficiently close to one. This process of (polynomially many) gradual changes, allows to transform the dummy matrix M_0 into a matrix M_t that is very close to M (and hence has a permanent that is very close to the permanent of M). Thus, approximately obtaining the parameters of M_t allows to approximate the permanent of M.

Exercises

Exercise 7.1 (enumeration problems) For any binary relation R, define the enumeration problem of R as a function $f_R : \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \cup \{\perp\}$ such that $f_R(x,i)$ equals the i^{th} element in $|R(x)|$ if $|R(x)| \geq i$ and $f_R(x,i) = \perp$ otherwise. The above definition refers to the standard lexicographic order on strings, but any other efficient order of strings will do.[11]

1. Prove that, for any polynomially bounded R, computing $\#R$ is reducible to computing f_R.

2. Prove that, for any $R \in \mathcal{PC}$, computing f_R is reducible to some problem in $\#\mathcal{P}$.

   Guideline: Consider the binary relation $R' = \{(\langle x, b\rangle, y) : (x,y) \in R \wedge y \leq b\}$, and show that f_R is reducible to $\#R'$. (Extra hint: Note that $f_R(x,i) = y$ if and only if $|R'(\langle x, y\rangle)| = i$ and for every $y' < y$ it holds that $|R'(\langle x, y'\rangle)| < i$.)

Exercise 7.2 (computing the permanent of integer matrices) Prove that computing the permanent of matrices with 0/1-entries is computationally equivalent to computing the number of perfect matchings in bipartite graphs.
(Hint: Given a bipartite graph $G = ((X,Y),E)$, consider the matrix M representing the edges between X and Y (i.e., the (i,j)-entry in M is 1 if the i^{th} vertex of X is connected to the j^{th} entry of Y), and note that only perfect matchings in G contribute to the permanent of M.)

Exercise 7.3 (computing the permanent modulo 3) Combining Proposition 7.8 and Theorem 7.16, prove that for every integer $n > 1$ that is relatively prime to c, computing the permanent modulo n is NP-hard under randomized reductions.[12] Since Proposition 7.8 holds for $c = 2^{10}$, hardness holds for every odd integer $n > 1$.

[11] An order of strings is a 1-1 and onto mapping μ from the natural numbers to the set of all strings. Such order is called efficient if both μ and its inverse are efficiently computable. The standard lexicographic order satisfies $\mu(i) = y$ if the (compact) binary expansion of i equals $1y$; that is $\mu(1) = \lambda$, $\mu(2) = 0$, $\mu(3) = 1$, $\mu(4) = 00$, etc.

[12] Actually, a sufficient condition is that n does not divide any power of c. Thus (referring to $c = 2^{10}$), hardness holds for every integer $n > 1$ that is not a power of 2. On the other hand, for any fixed $n = 2^e$, the permanent modulo n can be computed in polynomial-time [106, Thm. 3].

Guideline: Applying the reduction of Proposition 7.8 to the promise problem of deciding whether a 3CNF formula has a unique satisfiable assignment or is unsatisfiable. Use the fact that n does not divide any power of c.

Exercise 7.4 (negative values in Proposition 7.8) Assuming $\mathcal{P} \neq \mathcal{NP}$, prove that Proposition 7.8 cannot hold for a set I containing only non-negative integers. Note that the claim holds even if the set I is not finite (and even if I is the set of all non-negative integers).

Guideline: A reduction as in Proposition 7.8 provides a Karp-reduction of `3SAT` to deciding whether the permanent of a matrix with entries in I is non-zero. Note that the permanent of a *non-negative* matrix is non-zero if and only if the corresponding bipartite graph has a perfect matching.

Exercise 7.5 (error reduction for approximate counting) Show that the error probability δ in Definition 7.11 can be reduced from 1/3 (or even $(1/2) + (1/\mathrm{poly}(|x|))$ to $\exp(-\mathrm{poly}(|x|))$.

Guideline: Invoke the weaker procedure for an adequate number of times and take the *median* value among the values obtained in these invocations.

Exercise 7.6 (relative approximation for DNF satisfaction) Referring to the text of Section 7.2.1, prove the following claims.

1. Both assumptions regarding the general setting hold in case $S_i = C_i^{-1}(1)$, where $C_i^{-1}(1)$ denotes the set of truth assignments that satisfy the conjunction C_i.

   Guideline: In establishing the second assumption note that it reduces to the conjunction of the following two assumptions:

   (a) Given i, one can efficiently generate a uniformly distributed element of S_i. Actually, generating a distribution that is almost uniform over S_i suffices.

   (b) Given i and x, one can efficiently determine whether $x \in S_i$.

2. Prove Proposition 7.13, relating to details such as the error probability in an implementation of Construction 7.12.

3. Note that Construction 7.12 does not require exact computation of $|S_i|$. Analyze the output distribution in the case that we can only approximate $|S_i|$ up-to a factor of $1 \pm \varepsilon'$.

Exercise 7.7 (reducing the relative deviation in approximate counting) Prove that, for any $R \in \mathcal{PC}$ and every polynomial p and constant $\delta < 0.5$, there exists $R' \in \mathcal{PC}$ such that $(1/p, \delta)$-approximation for $\#R$ is reducible to $(1/2, \delta)$-approximation for $\#R'$.

Guideline: For $t(n) = \Omega(p(n))$, let $R' = \{(x, (y_1, ..., y_{t(|x|)})) : (\forall i)\,(x, y_i) \in R\}$. Note that $|R(x)| = |R'(x)|^{1/t(|x|)}$, and thus if $a = (1 \pm (1/2)) \cdot |R'(x)|$ then $a^{1/t(|x|)} = (1 \pm (1/2))^{1/t(|x|)} \cdot |R(x)|$. Furthermore, prove that $(1/p, \delta)$-approximation for $\#R$ is reducible to approximating $\#R''$ to within a factor of $F(n) = \exp(p(n))$ with error probability δ, for some $\mathbb{R}'' \in \mathcal{PC}$.
(Hint: Same as the main part. Note that the length of the solution for $R''(x)$ is larger than $p(|x|)$ and so there is nothing wrong in approximating $\#R''(|x|)$ to within $F(|x|)$.)

Exercise 7.8 (deviation reduction in approximate counting, cont.) In continuation to Exercise 7.7, prove that if R is NP-complete via parsimonious reductions then, for every positive polynomial p and constant $\delta < 0.5$, the problem of $(1/p, \delta)$-approximation for $\#R$ is reducible to $(1/2, \delta)$-approximation for $\#R$.
(Hint: Compose the reduction (to the problem of $(1/2, \delta)$-approximation for $\#R'$) provided in Exercise 7.7 with the parsimonious reduction of $\#R'$ to $\#R$.)
Prove that, for every function F' such that $F'(n) = \exp(n^{o(1)})$, we can also reduce the aforementioned problems to the problem of approximating $\#R$ to within a factor of F' with error probability δ.

Guideline: Using R'' as in Exercise 7.7, we encounter a technical difficulty. The issue is that the composition of the ("amplifying") reduction of $\#R$ to $\#R''$ with the parsimonious reduction of $\#R''$ to $\#R$ may increase the length of the instance. Indeed, the length of the new instance is polynomial in the length of the original instance, but this polynomial may depend on R'', which in turn depends on F'. Thus, we cannot use $F'(n) = \exp(n^{1/O(1)})$ but $F'(n) = \exp(n^{o(1)})$ is fine.

Exercise 7.9 Referring to the procedure in the proof Theorem 7.14, show how to use an NP-oracle in order to determine whether the number of solutions that "pass a random sieve" is greater than t. You are allowed queries of length polynomial in the length of x, h and in the size of t.
(Hint: Consider the set $S'_{R,H} \stackrel{\text{def}}{=} \{(x, i, h, 1^t) : \exists y_1, ..., y_t \text{ s.t. } \psi'(x, h, y_1, ..., y_t)\}$, where $\psi'(x, h, y_1, ..., y_t)$ holds if and only if the y_j are different and for every j it holds that $(x, y_j) \in R \wedge h(y_j) = 0^i$.)

Exercise 7.10 (parsimonious reductions and Theorem 7.16) Demonstrate the importance of parsimonious reductions in Theorem 7.16 by proving the following:

1. There exists a search problem $R \in \mathcal{PC}$ such that every problem in $\mathcal{PC}$ is reducible to R (by a non-parsimonious reduction) and still the the promise problem $(\mathtt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: Consider the following artificial witness relation R for $\mathtt{SAT}$ in which $(\phi, \sigma\tau) \in R$ if $\sigma \in \{0, 1\}$ and τ satisfies ϕ. Note that the standard witness relation of $\mathtt{SAT}$ is reducible to R, but this reduction is not parsimonious. Also note that $\mathtt{US}_R = \emptyset$ and thus $(\mathtt{US}_R, \overline{S}_R)$ is trivial.

2. There exists a search problem $R \in \mathcal{PC}$ such that $\#R$ is $\#\mathcal{P}$-complete and still the the promise problem $(\mathtt{US}_R, \overline{S}_R)$ is decidable in polynomial-time.

   Guideline: One easy proof is to use the relation suggested in the guideline to Part 1. A totally different proof relies on Theorem 7.7 and on the fact that it is easy to decide $(\mathtt{US}_R, \overline{S}_R)$ when R is the corresponding perfect matching relation (by computing the determinant).

Exercise 7.11 Prove that $\mathtt{SAT}$ is randomly reducible to deciding unique solution for $\mathtt{SAT}$, *without using the fact that* $\mathtt{SAT}$ *is NP-complete via parsimonious reductions.*

Guideline: Follow the proof of Theorem 7.16, while using the family of pairwise independent hashing functions provided in Construction 4.3. Note that, in this case, the condition $(\tau \in R_{\mathtt{SAT}}(\phi)) \wedge (h(\tau) = 0^i)$ can be directly encoded as a CNF formula. That is, consider the formula ϕ_h such that $\phi_h(z) \stackrel{\text{def}}{=} \phi(z) \wedge (h(z) = 0^i)$, and note that $h(z) = 0^i$ can be written as the conjunction of i clauses, where each clause is a CNF that is logically equivalent to the parity of some of the bits of z (where the identity of these bits is determined by h).

Exercise 7.12 (an alternative procedure for approximate counting) Adapt Step 1 of Construction 7.19 so to obtain an approximate counting procedure for $\#R$.

Guideline: For $m = 0, 1, ...\ell$, the procedure invokes Step 1 of Construction 7.19 until a negative answer is obtained, and outputs 2^m for the current value of m. For $|R(x)| > 1000\ell$, this yields a constant factor approximation of $|R(x)|$. In fact, we can obtain a better estimate by making additional queries at iteration m (i.e., queries of the form $(x, h, 1^i)$ for $i = 10\ell, ..., 1000\ell$). The case $|R(x)| \leq 1000\ell$ can be treated by using Step 2 of Construction 7.19, in which case we obtain an exact count.

Lecture 8

On Randomness Extractors

Extracting almost-perfect randomness from sources of weak (i.e., defected) randomness is crucial for the actual use of randomized algorithms, procedures and protocols. The latter are analyzed assuming that they are given access to a perfect random source, while in reality one typically has access only to sources of weak (i.e., highly imperfect) randomness. This gap is bridged by using randomness extractors, which are efficient procedures that (possibly with the help of little extra randomness) convert any source of weak randomness into an almost-perfect random source. Thus, randomness extractors are devices that greatly enhance the quality of random sources. In addition, randomness extractors are related to several other fundamental problems (see, e.g., Section 6.8.2 and [94]).

One key parameter, which was avoided in the foregoing abstract discussion, is the class of weak random sources from which we need to extract almost perfect randomness. Needless to say, it is preferable to make as little assumptions as possible regarding the weak random source. In other words, we wish to consider a wide class of such sources, and require that the randomness extractor (often referred to as the extractor) "works well" for any source in this class. A general class of such sources is defined in Section 8.1, but first we wish to mention that even for very restricted classes of sources no deterministic extractor can work.[1] To overcome this impossibility result, two approaches are used:

Seeded extractors: The first approach consists of considering randomized extractors that use a relatively small amount of randomness (in addition to the weak random source). That is, these extractors obtain two inputs: a short truly random seed and a relatively long sequence generated by an arbitrary source that belongs to the specified class of sources. This suggestion is motivated in two different ways:

1. The application may actually have access to an almost-perfect random source, but bits from this high-quality source are much more expensive than bits from the weak (i.e., low-quality) random source. Thus, it makes sense to obtain a few high-quality bits from the almost-perfect source and use them to "purify" the cheap bits obtained

[1] For example, consider the class of sources that output n-bit strings such that no string occurs with probability greater than $2^{-(n-1)}$ (i.e., twice its probability weight under the uniform distribution).

from the weak (low-quality) source. Thus, combining many cheap (but low-quality) bits with few high-quality (but expensive) bits, we obtain many high-quality bits.

2. In some applications (e.g., when using randomized algorithms), it may be possible to invoke the application multiple times, and use the "typical" outcome of these invocations (e.g., rule by majority in the case of a decision procedure). For such applications, we may proceed as follows: First we obtain an outcome r of the weak random source, then we invoke the application multiple times such that for every possible seed s we invoke the application feeding it with $\texttt{extract}(s,r)$, and finally we use the "typical" outcome of these invocations. Indeed, this is analogous to the context of derandomization (see [44, Sec. 8.3]), and likewise this alternative is typically not applicable to cryptographic and/or distributed settings.

Extraction from a few independent sources: The second approach consists of considering deterministic extractors that obtain samples from a few (say two) *independent* sources of weak randomness. Such extractors are applicable in any setting (including in cryptography), provided that the application has access to the required number of independent weak random sources.

In this chapter we focus on the first type of extractors (i.e., the *seeded extractors*). This choice is motivated by the closer connection between seeded extractors and other topics in the theory of computing. We also mention that our understanding of seeded extractors seem much more mature than the current state of knowledge regarding extraction from a few independent sources. Below we only present a definition that corresponds to the foregoing motivational discussion, and mention that its relation to other topics in complexity theory is discussed in Section 6.8.2 and in [94].

> Author's Note: *The style of the current text is quite laconic and various aspects of the main text are deferred to exercises. Hence, it is recommended to do all exercises of this chapter.*

8.1 Definitions and various perspectives

We first present a definition that corresponds to the foregoing motivational discussion, and later discuss its relation to other topics in complexity.

8.1.1 The Main Definition

A very wide class of weak random sources corresponds to sources in which no specific output is too probable. That is, the class is parameterized by a (probability) bound β and consists of all sources X such that for every x it holds that $\mathsf{Pr}[X = x] \leq \beta$. In such a case, we say that X has min-entropy[2] at least $\log_2(1/\beta)$. Indeed, we represent sources as random

[2] Recall that the entropy of a random variable X is defined as $\sum_x \mathsf{Pr}[X = x] \cdot \log_2(1/\mathsf{Pr}[X = x])$. Indeed the min-entropy of X equals $\min_x\{\log_2(1/\mathsf{Pr}[X = x])\}$, and is always upper-bounded by its entropy.

variables, and assume that they are distributed over strings of a fixed length, denoted n. An (n,k)-source is a source that is distributed over $\{0,1\}^n$ and has min-entropy at least k.

An interesting special case of (n,k)-sources is that of sources that are uniform over some subset of 2^k strings. Such sources are called (n,k)-flat. A useful observation is that *each (n,k)-source is a convex combination of (n,k)-flat sources*; that is, every (n,k)-source is obtained by picking a (n,k)-flat source with an appropriate probability distribution and producing a sample according to the picked source (see Exercise 8.1).

Definition 8.1 (extractor for (n,k)-sources):

1. *An algorithm* $\mathrm{Ext}:\{0,1\}^n\times\{0,1\}^d\to\{0,1\}^m$ *is called an* extractor with error ε for the class $\mathcal{C}$ *if for every source X in $\mathcal{C}$ it holds that* $\mathrm{Ext}(X,U_d)$ *is ε-close to U_m. If $\mathcal{C}$ is the class of (n,k)-sources, then* Ext *is called a (k,ε)-*extractor.

2. *An algorithm* Ext *is called a* strong extractor with error ε for $\mathcal{C}$ *if for every source X in $\mathcal{C}$ it holds that $(U_d,\mathrm{Ext}(X,U_d))$ is ε-close to (U_d,U_m). A* strong (k,ε)-extractor *is defined analogously.*

Using the aforementioned "decomposition" of (n,k)-sources into (n,k)-flat sources, it follows that Ext *is a (k,ε)-extractor if and only if it is an extractor with error ε for the class of (n,k)-flat sources.* (See Exercise 8.2, and note that a similar claim holds for strong extractors.) Thus, much of the technical analysis is conducted with respect to the class of (n,k)-flat sources. For example, by analyzing the case of (n,k)-flat sources it is easy to see that, for $d=\log_2(n/\varepsilon^2)+O(1)$, there exists a (k,ε)-extractor $\mathrm{Ext}:\{0,1\}^n\times\{0,1\}^d\to\{0,1\}^k$. (The proof employs the Probabilistic Method and uses a union bound on the (finite) set of all (n,k)-flat sources.)[3]

We seek, however, explicit extractors; that is, extractors that are implementable by polynomial-time algorithms. We note that the evaluation algorithm of any family of pairwise independent hash functions mapping n-bit strings to m-bit strings constitutes a (strong) (k,ε)-extractor for $\varepsilon=2^{-\Omega(k-m)}$ (see Theorem 4.5 (and Exercise 8.4)). However, these extractors necessarily use a long seed (i.e., $d\geq 2m$ must hold (and in fact $d=n+2m-1$ holds in Construction 4.3)). In Section 8.2 we survey constructions of efficient (k,ε)-extractors that obtain logarithmic seed length (i.e., $d=O(\log(n/\varepsilon))$).

On the importance of logarithmic seed length. The case of logarithmic seed length (i.e., $d=O(\log(n/\varepsilon))$) is of particular importance for a variety of reasons. First, when emulating a randomized algorithm using a defected random source (as in Item 2 of the motivational discussion of seeded extractors), the overhead is exponential in the length of the seed. Thus, the emulation of a generic probabilistic polynomial-time algorithm can be done in polynomial time only if the seed length is logarithmic. Similar considerations apply to other applications of extractors. Last, we note that logarithmic seed length is an absolute

[3]Indeed, the key fact is that the number of (n,k)-flat sources is $N\stackrel{\mathrm{def}}{=}\binom{2^n}{2^k}$. The probability that a random function $\mathrm{Ext}:\{0,1\}^n\times\{0,1\}^d\to\{0,1\}^k$ is not an extractor with error ε for a fixed (n,k)-flat source is upper-bounded by $p\stackrel{\mathrm{def}}{=}2^{2^k}\cdot\exp(-\Omega(2^{d+k}\varepsilon^2))$; see Exercise 8.3. Thus, for $d=\log_2(n/\varepsilon^2)+O(1)$ it holds that $N\cdot p\ll 1$. In fact, the same analysis applies to the extraction of $m=k+\log_2 n$ bits (rather than k bits).

lower-bound for (k,ε)-extractors, whenever $k < n - n^{\Omega(1)}$ (and the extractor is non-trivial (i.e., $m \geq 1$ and $\varepsilon < 1/2$)).

8.1.2 Extractors as averaging samplers

There is a close relationship between extractors and averaging samplers (which are defined towards the end of Section 6.7, see also Sections 6.6 and 6.8). We shall first show that any averaging sampler gives rise to an extractor. Let $G : \{0,1\}^n \to (\{0,1\}^m)^t$ be the sample generating algorithm of an averaging sampler having accuracy ε and error probability δ. That is, G uses n bits of randomness and generates t sample points in $\{0,1\}^m$ such that, for every $f : \{0,1\}^m \to [0,1]$ with probability at least $1-\delta$, the average of the f-values of these t pseudorandom points resides in the interval $[\overline{f} \pm \varepsilon]$, where $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)]$. Define $\mathrm{Ext} : \{0,1\}^n \times [t] \to \{0,1\}^m$ such that $\mathrm{Ext}(r,i)$ is the i^{th} sample generated by $G(r)$. We shall prove that Ext is a $(k, 2\varepsilon)$-extractor, for $k = n - \log_2(\varepsilon/\delta)$.

Suppose towards the contradiction that there exists a (n,k)-flat source X such that for some $S \subset \{0,1\}^m$ it is the case that $\mathsf{Pr}[\mathrm{Ext}(X,U_d) \in S] > \mathsf{Pr}[U_m \in S] + 2\varepsilon$, where $d = \log_2 t$ and $[t] \equiv \{0,1\}^d$. Define

$$B = \{x \in \{0,1\}^n : \mathsf{Pr}[\mathrm{Ext}(x,U_d) \in S] > (|S|/2^m) + \varepsilon\}.$$

Then, $|B| > \varepsilon \cdot 2^k = \delta \cdot 2^n$. Defining $f(z) = 1$ if $z \in S$ and $f(z) = 0$ otherwise, we have $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)] = |S|/2^m$. But, for every $r \in B$ the f-average of the sample $G(r)$ is greater than $\overline{f} + \varepsilon$, in contradiction to the hypothesis that the sampler has error probability δ (with respect to accuracy ε).

We now turn to show that extractors give rise to averaging samplers. Let $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ε)-extractor. Consider the sample generation algorithm $G : \{0,1\}^n \to (\{0,1\}^m)^{2^d}$ define by $G(r) = (\mathrm{Ext}(r,s))_{s \in \{0,1\}^d}$. We prove that G corresponds to an averaging sampler with accuracy ε and error probability $\delta = 2^{-(n-k-1)}$.

Suppose towards the contradiction that there exists a function $f : \{0,1\}^m \to [0,1]$ such that for $\delta 2^n = 2^{k+1}$ strings $r \in \{0,1\}^n$ the average f-value of the sample $G(r)$ deviates from $\overline{f} \stackrel{\text{def}}{=} \mathsf{E}[f(U_m)]$ by more than ε. Suppose, without loss of generality, that for at least half of these r's the average is greater than $\overline{f} + \varepsilon$, and let B denote the set of these r's. Then, for X that is uniformly distributed on B and is thus a (n,k)-source, we have

$$\mathsf{E}[f(\mathrm{Ext}(X,U_d))] > \mathsf{E}[f(U_m)] + \varepsilon,$$

which (using $|f(z)| \leq 1$ for every z) contradicts the hypothesis that $\mathrm{Ext}(X,U_d)$ is ε-close to U_m.

8.1.3 Extractors as randomness-efficient error-reductions

As may be clear from the foregoing discussion, extractors yield randomness-efficient methods for error-reduction. This is the case because *error-reduction is a special case of the sampling problem*, obtained by considering Boolean functions. Specifically, for a two-sided error decision procedure A, consider the function $f_x : \{0,1\}^{\rho(|x|)} \to \{0,1\}$ such that $f_x(r) = 1$ if $A(x,r) = 1$ and $f_x(r) = 0$ otherwise. Assuming that the probability that A is correct is at

least $0.5+\varepsilon$ (say $\varepsilon = 1/6$), error reduction amounts to providing a sampler with accuracy ε and any desired error probability $\delta \ll \varepsilon$ for the Boolean function f_x. Thus, by Section 8.1.2, any (k,ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{\rho(|x|)}$ with $k = n - \log(1/\delta) - 1$ yields the desired error-reduction, provided that 2^d is feasible (e.g., $2^d = \mathrm{poly}(\rho(|x|))$, where $\rho(\cdot)$ represents the randomness complexity of the original algorithm A). The question of interest here is how does n (which represents the randomness complexity of the corresponding sampler) grow as a function of $\rho(|x|)$ and δ.

Error-reduction using the extractor $\mathrm{Ext}: \{0,1\}^n \to \{0,1\}^{\rho(|x|)} \times [\mathrm{poly}(\rho(|x|))]$

	error probability	randomness complexity
original algorithm	1/3	$\rho(\|x\|)$
resulting algorithm	δ (may depend on $\|x\|$)	n (function of $\rho(\|x\|)$ and δ)

Needless to say, the answer to the foregoing question depends on the quality of the extractor that we use. In particular, using Part 1 of the forthcoming Theorem 8.3, we note that for every $\alpha > 1$, one can obtain $n = O(\rho(|x|)) + \alpha \log_2(1/\delta)$, for any $\delta > 2^{-\mathrm{poly}(\rho(|x|))}$. Note that, for $\delta < 2^{-O(\rho(|x|))}$, this bound on the randomness-complexity of error-reduction is better than the bound of $n = \rho(|x|) + O(\log(1/\delta))$ that is provided (for the reduction of one-sided error) by the Expander Random Walk Generator (of Section 3.4), albeit the number of samples here is larger (i.e., $\mathrm{poly}(\rho(|x|)/\delta)$ rather than $O(\log(1/\delta))$).

Mentioning the reduction of *one-sided* error-probability brings us to a corresponding relaxation of the notion of an extractor, which is called a disperser. Loosely speaking, a (k,ε)-disperser is only required to hit (with positive probability) any set of density greater than ε in its image, rather than produce a distribution that is ε-close to uniform.

Definition 8.2 (dispersers): *An algorithm* $\mathrm{Dsp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is called a* (k,ε)-disperser *if for every* (n,k)*-source* X *the support of* $\mathrm{Dsp}(X,U_d)$ *covers at least* $(1-\varepsilon)\cdot 2^m$ *points. Alternatively, for every set* $S \subset \{0,1\}^m$ *of size greater than* $\varepsilon 2^m$ *it holds that* $\mathsf{Pr}[\mathrm{Dsp}(X,U_d) \in S] > 0$.

Dispersers can be used for the reduction of one-sided error analogously to the use of extractors for the reduction of two-sided error. Specifically, regarding the aforementioned function f_x (and assuming that $\mathsf{Pr}[f_x(U_{\ell(|x|)}) = 1] > \varepsilon$), we may use any (k,ε)-disperser $\mathrm{Dsp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{\ell(|x|)}$ towards finding a point z such that $f_x(z) = 1$. Indeed, if $\mathsf{Pr}[f_x(U_{\ell(|x|)}) = 1] > \varepsilon$ then there are less than 2^k points z such that $(\forall s \in \{0,1\}^d)\, f_x(\mathrm{Dsp}(z,s)) = 0$, and thus the one-sided error can be reduced from $1-\varepsilon$ to $2^{-(n-k)}$ while using n random bits. (Note that dispersers are closely related to hitters (cf. Section 6.9), analogously to the relation of extractors and averaging samplers.)

8.1.4 Other perspectives

Extractors and dispersers have an appealing interpretation in terms of bipartite graphs. Starting with dispersers, we view any (k,ε)-disperser $\mathrm{Dsp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ as a bipartite graph $G = ((\{0,1\}^n, \{0,1\}^m), E)$ such that $E = \{(x, \mathrm{Dsp}(x,s)) : x \in \{0,1\}^n, s \in \{0,1\}^d\}$. This graph has the property that *any* subset of 2^k vertices on the left (i.e., in $\{0,1\}^n$) has a neighborhood that contains at least a $1-\varepsilon$ fraction of the vertices of the right, which is remarkable in the typical case where d is small (e.g., $d = O(\log n/\varepsilon)$)

and $n \gg k \geq m$ whereas $m = \Omega(k)$ (or at least $m = k^{\Omega(1)}$). Furthermore, if Dsp is efficiently computable then this bipartite graph is strongly constructible in the sense that, given a vertex on the left, one can efficiently find each of its neighbors. Any (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ yields an analogous graph with an even stronger property: the neighborhood multi-set of *any* subset of 2^k vertices on the left covers the vertices on the right in an almost uniform manner.

An odd perspective. In addition to viewing extractors as averaging samplers, which in turn may be viewed within the scope of the pseudorandomness paradigm, we mention here an even more odd perspective. Specifically, randomness extractors may be viewed as randomized algorithms (distinguishers) designed on purpose such that to be fooled by any weak random source (but not by an even worse source). Specifically, for any (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $\varepsilon \leq 1/100$, $m = k = \omega(\log n/\varepsilon)$ and $d = O(\log n/\varepsilon)$, consider the following class of distinguishers (or tests), parameterized by subsets of $\{0,1\}^m$: for $S \subset \{0,1\}^m$, the test T_S satisfies $\mathsf{Pr}[T_S(x)=1] = \mathsf{Pr}[\mathrm{Ext}(x, U_d) \in S]$ (i.e., on input $x \in \{0,1\}^n$, the test uniformly selects $s \in \{0,1\}^d$ and outputs 1 if and only if $\mathrm{Ext}(x,s) \in S$). Then, as shown next, any (n,k)-source is "pseudorandom" with respect to this class of distinguishers, but sufficiently "non-(n,k)-sources" are not "pseudorandom" with respect to this class of distinguishers.

1. For every (n,k)-source X and every $S \subset \{0,1\}^m$, the test T_S does not distinguish X from U_n (i.e., $\mathsf{Pr}[T_S(X)=1] = \mathsf{Pr}[T_S(U_n)=1] \pm 2\varepsilon$), because $\mathrm{Ext}(X, U_d)$ is 2ε-close to $\mathrm{Ext}(U_n, U_d)$ (since each is ε-close to U_m).

2. On the other hand, for every $(n, k-d-4)$-flat source Y there exists a set S such that T_S distinguish Y from U_n with gap at least 0.9 (e.g., for S that equals the support of $\mathrm{Ext}(Y, U_d)$, it holds that $\mathsf{Pr}[T_S(Y)=1] = 1$ but $\mathsf{Pr}[T_S(U_n)=1] \leq \mathsf{Pr}[U_m \in S] + \varepsilon = 2^{d+(k-d-4)-m} + \varepsilon < 0.1$). Furthermore, any source that has entropy below $(k/4) - d$ will be detected as defected by this class (with probability at least 2/3).[4]

Thus, this weird class of tests deems each (n,k)-source as "pseudorandom" while deeming sources of significantly lower entropy (e.g., entropy lower than $(k/4) - d$) as non-pseudorandom. Indeed, this perspective stretches the pseudorandomness paradigm quite far.

8.2 Constructions

Recall that we seek explicit constructions of extractors; that is, functions $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ that can be computed in polynomial-time. The question, of course, is of parameters; that is, having explicit (k, ε)-extractors *with m as large as possible and d as small as possible*. We first note that, except for "pathological" cases[5], both $m \leq k + d -$

[4] For any such source Y, the distribution $Z = \mathrm{Ext}(Y, U_d)$ has entropy at most $k/4 = m/4$, and thus is 0.7-far from U_m (and 2/3-far from $\mathrm{Ext}(U_n, U_d)$). The lower-bound on the statistical distance between Z and U_m can be proved by the contrapositive: if Z is δ-close to U_m then its entropy is at least $(1-\delta) \cdot m - 1$ (e.g., by using Fano's inequality, see [34, Thm. 2.11.1]).

[5] That is, for $\varepsilon < 1/2$ and $m > d$.

$(2\log_2(1/\varepsilon)-O(1))$ and $d \geq \log_2((n-k)/\varepsilon^2)-O(1)$ must hold, regardless of the explicitness requirement. The aforementioned bounds are in fact tight; that is, there exist (non-explicit) (k,ε)-extractors with $m = k+d-2\log_2(1/\varepsilon)-O(1)$ and $d = \log_2((n-k)/\varepsilon^2)+O(1)$. The obvious goal is meeting these bounds via explicit constructions.

8.2.1 Some known results

Despite tremendous progress on this problem (and occasional claims regarding "optimal" explicit constructions), the ultimate goal has not yet been reached. Nevertheless, the known explicit constructions are pretty close to being optimal.

Theorem 8.3 (explicit constructions of extractors): *Explicit (k,ε)-extractors of the form $\mathrm{Ext}:\{0,1\}^n\times\{0,1\}^d\to\{0,1\}^m$ exist for the following cases (i.e., settings of the parameters d and m):*

1. *For $d = O(\log n/\varepsilon)$ and $m = (1-\alpha)\cdot(k-O(d))$, where $\alpha > 0$ is an arbitrarily small constant and provided that $\varepsilon > \exp(-k^{1-\alpha})$.*
2. *For $d = (1+\alpha)\cdot\log_2 n$ and $m = k/\mathrm{poly}(\log n)$, where $\varepsilon,\alpha > 0$ are arbitrarily small constants.*

Proofs of Part 1 and Part 2 can be found in [55] and [95], respectively. We note that, for the sake of simplicity, we did not quote the best possible bounds. Furthermore, we did not mention additional incomparable results (which are relevant for different ranges of parameters).

We refrain from providing an overview of the proof of Theorem 8.3, but rather review (next) the conceptual insight that underlies many of the results that belong to the current "generation" of constructions.

8.2.2 Advanced Topic: The pseudorandomness connection

The connection between extractors and certain pseudorandom generators, discovered by Trevisan [104], is the starting point of the current generation of constructions of extractors. This connection is surprising because it went in a non-standard direction; that is, transforming certain pseudorandom generators into extractors. We note that computational objects are typically more complex than the corresponding information theoretical objects (cf. e.g., [44, Chap. 7]). Thus, if pseudorandom generators and extractors are at all related (which was not suspected before [104]), then this relation should not be expected to help in the construction of extractors, which seem to be information theoretic objects. Nevertheless, the discovery of this relation did yield a breakthrough in the study of extractors.[6]

But before describing the connection, let us wonder for a moment. Just looking at the syntax, we note that pseudorandom generators have a single input (i.e., the seed), while extractors have two inputs (i.e., the n-bit long source and the d-bit long seed). But taking a second look at the Nisan–Wigderson Generator (i.e., the combination of [44, Const 8.17] with an amplification of worst-case to average-case hardness), we note that this construction

[6] We note that once the connection became better understood, influence started going in the "right" direction: from extractors to pseudorandom generators.

can be viewed as taking two inputs: a d-bit long seed and a "hard" predicate on d'-bit long strings (where $d' = \Omega(d)$).[7] Now, an appealing idea is to use the n-bit long source as a (truth-table) description of a (worst-case) hard predicate (which indeed means setting $n = 2^{d'}$). The key observation is that *even if the source is only weakly random, then it is likely to represent a predicate that is inapproximable* (as in the hypothesis of [44, Thm. 8.18]). Indeed, the following text relies on close familiaity with the Nisan–Wigderson Generator and its analysis ([44, Const 8.17] and [44, Thm 8.18], respectively).

Recall that the aforementioned construction is supposed to yield a pseudorandom generator whenever it starts with a hard predicate. In the current context, where there are no computational restrictions, pseudorandomness is supposed to hold against any (computationally unbounded) distinguisher, and thus here pseudorandomness means being statistically close to the uniform distribution (on strings of the adequate length, denoted ℓ). Intuitively, this makes sense only if the observed sequence is shorter than the amount of randomness in the source (and seed), which is indeed the case (i.e., $\ell < k + d$, where k denotes the min-entropy of the source). Hence, there is hope to obtain a good extractor this way.

To turn the hope into reality, we need a proof (which is sketched next). Looking again at the Nisan–Wigderson Generator, we note that the proof of indistinguishability of this generator provides a black-box procedure for approximating the underlying predicate when given oracle access to any potential distinguisher. Specifically, in the proofs of [44, Thm 8.18] (which holds for any $\ell = 2^{\Omega(d')}$)[8], this black-box procedure was implemented by a *relatively small circuit* (which depends on the underlying predicate). Hence, this procedure contains relatively little information (regarding the underlying predicate), on top of the observed ℓ-bit long output of the extractor/generator. Specifically, for some fixed polynomial p, the amount of information encoded in the procedure (and thus available to it) is upper-bounded by $p(\ell)$, while the procedure is supposed to approximate the underlying predicate in the sense that this approximation determines a set of at most $p(\ell)$ predicates that contain the original predicate. Thus, $b = p(\ell)^2$ bits of information are supposed to fully determine the underlying predicate, which in turn is identical to the n-bit long source. However, if the source has min-entropy exceeding b, then it cannot be fully determined using only b bits of information.

It follows that the foregoing construction constitutes a $(b + O(1), 1/6)$-extractor (outputting $\ell = b^{\Omega(1)}$ bits), where the constant $1/6$ is the one used in the proof of [44, Thm 8.18] (and the argument holds provided that $b = n^{\Omega(1)}$). Note that this extractor uses a seed of length $d = O(d') = O(\log n)$. The argument can be extended to obtain $(k, \mathrm{poly}(1/k))$-extractors that output $k^{\Omega(1)}$ bits using seeds of length $d = O(\log n)$, provided that $k = n^{\Omega(1)}$.

We stress that the foregoing description has only referred to two abstract properties of the Nisan–Wigderson Generator: (1) the fact that this generator uses any worst-case hard predicate as a black-box, and (2) the fact that its analysis uses any distinguisher as a black-box. In particular, we viewed the amplification of worst-case hardness to inapproximability (performed in [44, Const 7.19]) as part of the construction of the pseudorandom generator. An alternative presentation, which is more self-contained, replaces the amplification step

[7] Indeed, to fit the current context, we have modified some notation. In [44, Const 8.17] the length of the seed is denoted by k and the length of the input for the predicate is denoted by m.

[8] Recalling that $n = 2^{d'}$, the restriction $\ell = 2^{\Omega(d')}$ implies $\ell = n^{\Omega(1)}$.

of [44, Const 7.19] by a direct argument in the current (information theoretic) context and plugs the resulting predicate directly into [44, Const 8.17]. The advantages of this alternative include using a simpler amplification (since amplification is simpler in the information theoretic setting than in the computational setting), and deriving transparent construction and analysis (which mirror [44, Const 8.17] and [44, Thm 8.18], respectively).

The alternative presentation. The foregoing analysis transforms a generic distinguisher into a procedure that computes the underlying predicate correctly on each input, which fully determines this predicate. Hence, an upper-bound on the information available to this procedure yields an upper-bound on the number of possible outcomes of the source that are bad for the extractor. In the alternative presentation, we transforms a generic distinguisher into a procedure that only approximates the underlying predicate; that is, the procedure yields a function that is relatively close to the underlying predicate. If the potential underlying predicates are far apart, then this yields the desired bound (on the number of bad source-outcomes that correspond to such predicates). Thus, the idea is to encode the n-bit long source by an error correcting code of length $n' = \mathrm{poly}(n)$ and relative distance $0.5-(1/n)^2$, and use the resulting codeword as a truth-table of a predicate for [44, Const 8.17].[9] Such codes (coupled with efficient encoding algorithms) do exist (see §2.3.2.5), and the benefit in using them is that each n'-bit long string (determined by the information available to the aforementioned approximation procedure) may be $(0.5-(1/n))$-close to at most $O(n^2)$ codewords[10] (which correspond to potential predicates). Thus, each approximation procedure rules out at most $O(n^2)$ potential predicates (i.e., source outcomes). In summary, *the resulting extractor converts the n-bit input x into a codeword $x' \in \{0,1\}^{n'}$, viewed as a predicate over $\{0,1\}^{d'}$* (where $d' = \log_2 n'$), *and evaluates this predicate at the ℓ projections of the d-bit long seed, where these projections* (to d' bits) *are determined by the corresponding set system* (i.e., the ℓ-long sequence of d'-subsets of $[d]$ that is used in [44, Const 8.17]). The analysis mirrors the proof of [44, Thm 8.18] and yields a bound of $2^{O(\ell^2)} \cdot O(n^2)$ on the number of bad outcomes for the source, where $O(\ell^2)$ upper-bounds the amount of information encoded in (and available to) the approximation procedure, and $O(n^2)$ upper-bounds the number of source-outcomes that correspond to codewords that are each $(0.5-(1/n))$-close to any fixed approximation procedure.

Notes

The interested reader is referred to a survey of Shaltiel [94]. This survey contains a comprehensive introduction to the area, including an overview of the ideas that underly the various constructions. In particular, the survey describes the approaches used before the discovery of the pseudorandomness connection, the connection itself (and the constructions that arise from it), and the "third generation" of constructions that followed.

The aforementioned survey predates the most recent constructions (of extractors) that extract a constant fraction of the min-entropy using a logarithmically long seed (cf. Part 1 of Theorem 8.3). Such constructions were first presented in [78] and improved (using different

[9] Indeed, the use of this error correcting code replaces the hardness-amplification step of [44, Const 7.19].
[10] See Appendix 2.3.4.

ideas) in [55]. Indeed, we refer to reader to [55], which provides a self-contained description of the best known extractor (for almost all settings of the relevant parameters).

Extractors for more restricted classes of sources. Recall that a random seed is essential for extraction even when the class of sources is confined to $(n, n-1)$-sources. In contrast to this fact, deterministic (i.e., seedless) extraction is possible for several restricted classes of sources; in fact, one may view the problem of (deterministic) extraction from few independent sources as a special case of extraction from a restricted classes of sources (i.e., the class of sources that consist of few parts such that the distribution on each part is independent of the distribution on the other parts).

Exercises

Exercise 8.1 (convex combination of (n,k)-flat sources) Prove that for every natural number K, each $(n, \log K)$-source X is a convex combination of $(n, \log K)$-flat sources; that is, there exists a sequence of non-negative numbers, $(\alpha_S)_{S \subseteq \{0,1\}^n : |S|=K}$, such taht for every $x \in \{0,1\}^n$ it holds that $\mathsf{Pr}[X=x] = \sum_{S:|S|=K} \alpha_S \cdot \mathsf{Pr}[X_S=x]$, where X_S is uniform over S.

Guideline: Observe first that any convex combination of any (two) $(n, \log K)$-sources yields a $(n, \log K)$-source. Next view each $(n, \log K)$-source as a 2^n-dimentional vector over $\mathbb{R}$ (i.e., the $(n, \log K)$-source X is represented by the vector $(\mathsf{Pr}[X=x])_{x \in \{0,1\}^n}$). Show that the $(n, \log K)$-sources constitute a convex polytope in $\mathbb{R}^{2^n}$, and that the vertices of this polytope are $(n, \log K)$-flat sources.

Exercise 8.2 (extraction for (n,k)-flat sources) Prove that $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor if and only if it is an extractor with error ε for the class of (n,k)-flat sources. Prove the same for strong extractors.

Guideline: Let X and the α_S and X_S's be as in Exercise 8.1. Then, for every $s \in \{0,1\}^d$ and $v \in \{0,1\}^m$ it holds that $\mathsf{Pr}[\mathrm{Ext}(X,s)=v] = \sum_S \alpha_S \cdot \mathsf{Pr}[\mathrm{Ext}(X_S,s)=v]$.

Exercise 8.3 (on the existence of extractors) Prove that, for $d = \log_2(n/\varepsilon^2) + O(1)$ and $m = k + \log_2 n$, there exists a (k,ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$.

Guideline: By Exercise 8.2, it suffices to show that an extractor with error ε for the class of (n,k)-flat sources. The proof employs the Probabilistic Method and uses a union bound, while relying on the fact that the number of (n,k)-flat sources is $N \stackrel{\text{def}}{=} \binom{2^n}{2^k}$. The probability that a random function $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is not an extractor with error ε for a fixed (n,k)-flat source is upper-bounded by $p \stackrel{\text{def}}{=} 2^{2^m} \cdot \exp(-\Omega(2^{d+k}\varepsilon^2))$, because p bounds the probability that when selecting 2^{d+k} random k-bit long strings there exists a set $T \subset \{0,1\}^m$ that is hit by more than $((|T|/2^m) + \varepsilon) \cdot 2^{d+k}$ of these strings. Note that for a sufficiently large constant c, letting $d = \log_2(n/\varepsilon^2) + c$, it holds that $N \cdot p < 2^{n \cdot 2^k + 2^m - \Omega(2^{d+k}\varepsilon^2)} \ll 1$, since $2^{d+k}\varepsilon^2 = 2^c \cdot n \cdot 2^k$ whereas $n \cdot 2^k + 2^m = 2n2^k$.

Exercise 8.4 (pairwise independent hashing as extractors) Show that the evaluation algorithm of any family of pairwise independent hash functions mapping n-bit strings to m-bit strings constitutes a strong $(k, 2^{-(k-m-3)/3})$-extractor.

Guideline: See Theorem 4.5.

Lecture 9

A Taste of Randomized Computations

The purpose of this text is to demonstrate the usage of randomization in a variety of computational settings. Our choice is governed by the desire to focus on the randomization aspect of the solution and avoid any complicated details that are due to other aspects of the computational problem. Thus, we avoid any example that requires substantial problem-specific background. Our examples are grouped in three (subjective) categories:

1. Traditional algorithmic problems. Here we consider *randomized algorithms* for graph theoretic problems such as finding a perfect matching, algebraic problems such as testing polynomial identity, and approximation problems such as approximating the number of satisfying assignments to a DNF formula.

2. Traditional complexity questions. Here we present results such as the *randomized reductions* of Approximate Counting to $\mathcal{NP}$, and of SAT to unique-SAT.

3. Distributed and Parallel Computing. Here we consider *randomized procedures* for *distributed tasks* such as Testing String Equality, Byzantine Agreement, and routing in networks.

We stress that our presentation is merely aimed at *demonstrating* the usage of randomization, and that no attempt was made to present a coherent theory of randomized computation. Furthermore, our presentation tends to be laconic (i.e., it lacks some technical details as well as wider perspective).[1]

We mention that the interplay between randomness and computation is one of the most fascinating scientific phenomena uncovered in the last couple of decades. This interplay is at the heart of modern cryptography and plays a fundamental role in complexity theory at large. Specifically, the interplay of randomness and computation is pivotal to several intriguing notions of probabilistic proof systems and is the focal of the computational approach to randomness. In this text, we have avoided the above areas. For an introduction to to

[1] We also mention that two of the examples have appeared in other lectures, but the presentation here is somewhat different.

these three, somewhat interwoven areas, the interested reader is referred to the text *Modern Cryptography, Probabilistic Proofs and Pseudorandomness* [41] (or to the corresponding chapters in [44]). For a more systematic, detailed and inclusive exposition of Randomized Algorithms, the interested reader is referred to a textbook by Motwani and Raghavan [83].

9.1 Randomized Algorithms

Conspicuous omissions in this category include some of the most well-known randomized algorithms (e.g., many in the domain of computational number theory), as well as the Markov Chain approach to approximate counting. As stated above, the reason for these omissions is that these algorithms either require specialized (and unrelated to randomness) background or are quite involved to present and/or analyze.

9.1.1 Approximate Counting of DNF satisfying assignments or, a twist on naive sampling

The problem considered here is to approximate the number of satisfying assignment to a DNF formula up-to a *constant factor*. We note that given ε and oracle access to any function $f : \{0,1\}^n \to \{0,1\}$, it is easy to approximate the fraction $|\{x : f(x) = 1\}|/2^n$ up-to an ε *additive deviation*. Specifically, a sample of $O(\varepsilon^{-2}\log(1/\delta))$ points has average value that, with probability at least $1-\delta$, is at most ε-away from the correct value. However, our aim is to provide relative (rather than absolute) approximation of this fraction (i.e., given $\varepsilon > 0$ the task is to approximate the above fraction up-to a $1 \pm \varepsilon$ factor).

Let $\varphi = \bigvee_{i=1}^{m} C_i$, where $C_i : \{0,1\}^n \to \{0,1\}$ is a conjunction, be a DNF formula. Actually, we will deal with the more general problem in which we are given (implicitly) m subsets $S_1, ..., S_m \subseteq \{0,1\}^n$ and wish to approximate $|\bigcup_i S_i|$. In our case S_i will be the set of assignments satisfying the conjunction C_i. We make several computational assumptions regarding these sets (letting efficient mean implementable in time polynomial in $n \cdot m$):

1. Given i and x, one can efficiently determine whether or not $x \in S_i$.
2. Given i, one can efficiently determine $|S_i|$.
3. Given i, one can efficiently generate a uniformly distributed element of S_i.

These assumptions are clearly satisfied in the case $S_i = C_i^{-1}(1)$ considered above. The key observation is that

$$\left|\bigcup_{i=1}^{m} S_i\right| = \sum_{i=1}^{m} \left|S_i \setminus \bigcup_{j<i} S_j\right| \tag{9.1}$$

$$= \sum_{i=1}^{m} |S_i| \cdot \mathsf{Pr}_{s\in S_i}\left[s \notin \bigcup_{j<i} S_j\right] \tag{9.2}$$

and that the probabilities in Eq. (9.2) can be approximated up-to ε' (with overwhelming success probability) by taking $\mathrm{poly}(n/\varepsilon')$ many samples. This leads to the following algorithm

Algorithm: On input parameters ε and δ, set $\varepsilon' = \varepsilon/m$ and $\delta' = \delta/m$. For $i = 1$ to m do

1. Let $p_i \stackrel{\text{def}}{=} \mathsf{Pr}_{s\in S_i}[s \notin \bigcup_{j<i} S_j]$. Using a sample of size $t \stackrel{\text{def}}{=} O((1/\varepsilon')^2 \log(1/\delta'))$, approximate p_i by $\widetilde{p}_i$ so that $\mathsf{Pr}[|\widetilde{p}_i - p_i| > \varepsilon'] < \delta'$. That is, we uniformly select t samples in S_i, and test for each sample whether or not it resides in $\bigcup_{j<i} S_j$.

2. Compute $|S_i|$, and let $a_i \stackrel{\text{def}}{=} \widetilde{p}_i \cdot |S_i|$.

Output the sum of the a_i's.

Analysis: Let $N_i = p_i \cdot |S_i|$. We are interested in the quality of the approximation to $\sum_i N_i$ provided by $\sum_i a_i$. With probability at least $1 - m\cdot\delta'$, we have $a_i = (p_i \pm \varepsilon')\cdot|S_i| = N_i \pm \varepsilon'\cdot|S_i|$, for all i's, and so $\sum_i a_i = \sum_i N_i \pm \varepsilon' \cdot \sum_i |S_i|$. However, $\max_i(|S_i|) \leq |\bigcup_i S_i| = \sum_i N_i$, and so

$$\begin{aligned}\sum_{i=1}^{m} a_i &= \sum_{i=1}^{m} N_i \pm m \cdot \varepsilon' \cdot \max_{1\leq i\leq m} |S_i| \\ &= (1 \pm m\varepsilon') \cdot \sum_{i=1}^{m} N_i = (1 \pm \varepsilon) \cdot \sum_{i=1}^{m} N_i\end{aligned}$$

Note that the above approach does not require exact computation of $|S_i|$, nor exact uniform selection in S_i. Instead, ability to approximate $|S_i|$ up-to a factor of $1 \pm \varepsilon'$ within time related to $\text{poly}(n/\varepsilon')$ suffices. Likewise, it suffice to generate in time related to $\text{poly}(n/\varepsilon')$ a distribution that is at most ε'-away from the uniform distribution over S_i.

The algorithm presented above is actually a deterministic reduction of the task of approximating the size of one set (in the relative sense) to the task of providing absolute approximations to some fractions. It utilizes the hypothesis that the first set can be expressed as a union of feasibly many sets for which certain natural operations (e.g., deciding membership, approximating the size) can be performed efficiently. Thus, this approach may be applicable to some sets, but not to their complement. We stress that, in general, relative approximation may be feasible for one quantity, but not for its complement (e.g., it is NP-Hard to approximate the number of UNsatisfying assignment to a DNF formula up-to any factor).

9.1.2 Finding a perfect matching or, on the loneliness of the extremum

The problem considered here is to find a perfect matching in a graph. The specific goal is to obtain a fast parallel algorithm, which is the reason we do not follow the standard combinatorial approach (of iteratively augmenting the current matching using alternating paths). Instead, we rely on the following Isolation Lemma that asserts that when assigning each edge a random weight, taken from a sufficiently large domain, there is a unique perfect matching of minimum (resp., maximum) weight. The lemma extends to arbitrary set systems.

Lemma 9.1 (The Isolation Lemma): *Let $S_1, S_2, ..., S_t \subseteq [m] \stackrel{\text{def}}{=} \{1, 2, ..., m\}$ be distinct sets, and let $w_1, w_2, ..., w_m$ be independently and uniformly chosen in $[2m]$. Then, with probability at least $1/2$, there exists a unique j so that $\sum_{i \in S_j} w_i$ equals $\min_{k \in [t]}(\sum_{i \in S_k} w_i)$.*

In our application $[m]$ corresponds to the set of edges, and the S_i's to perfect matchings in the graph.

Proof: For $i = 1, ..., m$, consider the event E_i defined as the existence of two sets (i.e., S_j's) with minimum weight so that one set contains i and the other set does not contain i. It suffices to show that the probability that E_i occurs is at most $1/2m$. The latter is proven by considering a random process in which the weight of i (i.e., w_i) is selected last.

Suppose that the values of all other w_j's (with $j \neq i$) have already been determined. Let S^- be a set of minimum weight among all sets not containing i, and w^- be its weight (i.e., $w^- \stackrel{\text{def}}{=} \min_{j: i \notin S_j}(\sum_{k \in S_j} w_k)$). Similarly, let S^+ be a set of minimum weight among all sets obtained by omitting i from sets that contain it, and w^+ be its weight (i.e., $w^+ \stackrel{\text{def}}{=} \min_{j: i \in S_j}(\sum_{k \in S_j \setminus \{i\}} w_k)$). Then, event E_i occurs if and only if $w^- = w^+ + w_i$, which happens with probability $1/2m$ if $(w^- - w^+) \in [2m]$, and with probability 0 otherwise. ■

Algorithm: On input a bipartite graph $G = (U, V, E)$, do:

1. For each edge $e \in E$, uniformly and independently select a weight $w_e \in [2m]$, where $m \stackrel{\text{def}}{=} |E|$.

2. Try to compute the value of the minimum-weight perfect-matching. This is done by computing the determinant of the matrix, denoted A, obtained by setting the (u, v)-entry to 2^{w_e} if $e = (u, v)$ and to 0 if $(u, v) \notin E$. In case the determinant is 0, halt stating that the graph has no perfect matching. Otherwise, the value of the minimum-weight perfect-matching is set to be the largest i so that the value of the determinant is divisible by 2^i. (The determinant can be computed by a fast parallel algorithm.)

3. For each $e \in E$, try to compute the value of the minimum-weight perfect-matching among those not containing the edge e. This is done (as above) by computing the determinant of the matrix, denoted A_e, obtained from A by resetting the e-entry to 0. All these computations can be conducted in parallel.

4. A candidate perfect matching is retrieved by including all edges e for which the value (of the min-weight perfect matching) found in Step 3 is different than the one found in Step 2.

The algorithm for general graphs is a variation of the above (and is not described here). Note that Steps 1 and 2 (by themselves) provide a randomized algorithm for determining whether a bipartite graph has a perfect matching.

Analysis: The determinant of A sums (possibly with minus sign) the contributions of all perfect matchings in the graph G, where the contribution of a perfect matching M equals $\pm\prod_{e\in M} 2^{w_e} = \pm 2^{\sum_{e\in M} w_e}$. We may assume that the graph has a perfect matching, or else the determinant computed in Step 2 is 0. Assume that the weights (i.e., w_e's) are such that there exists a unique perfect matching of minimum weight. Denote this (minimum-weight perfect) matching by M, and denote its weight by W. In such a case, the determinant of A is of the form $2^W + r \cdot 2^{W+1}$, where r is an integer (possibly zero). This is so because the contribution of the *unique* minimum-weight perfect-matching is $\pm 2^W$, and the contribution of each other perfect-matching is $\pm 2^{W'}$, where $W' > W$ (are both integers). Likewise, for every edge e not in M, the determinant of A_e is of the form $2^W + r \cdot 2^{W+1}$, where again r is an integer. On the other hand, for every edge e in M, the determinant of A_e is either zero or $r \cdot 2^{W+1}$, with r being a non-zero integer.

Advanced Topic: A Parenthetical Comment[2]

It is tempting to think that when selecting weights as above, the minimum-weight perfect matching may be uniformly distributed among all perfect matchings. As shown below, this is not always the case (which is unfortunate, because otherwise we would have obtained a simple probabilistic polynomial-time algorithm for uniformly generating a perfect matching in a graph).

Consider a graph in which the set of perfect matchings consists of two types of matchings. There are 2^n matchings of the first type, a generic one having the form $\{e_{2i-\sigma_i} : i = 1, ..., n\}$, where $\sigma_1, ..., \sigma_n \in \{0, 1\}$. In addition, there is a single matching of the second type, denoted $\{e_{2n+i} : i = 1, ..., n\}$. We claim that the probability that the minimum-weight perfect matching is a specific matching of the first type is exponentially smaller than the probability that the minimum-weight perfect matching is the matching of the second type. This claim holds for weights distributed as above, as well as for several other distributions (e.g., the Normal Distribution). For sake of simplicity, we consider weights uniformly distributed in the interval $[0, 1]$. The claim is proven by combining the following two facts.

Fact 9.2 *With overwhelmingly high probability, the value of the minimum-weight matching among all 2^n matchings of the first type is* at least *cn, where c is any constant smaller than* $1/3$ (e.g., $c = 0.32$).

Proof Sketch: This follows by observing that

$$\min_{\sigma_1,...,\sigma_n\in\{0,1\}} \left(\sum_{i=1}^{n} w_{2i-\sigma_i} \right) = \sum_{i=1}^{n} \min(w_{2i-1}, w_{2i})$$

and that the expected value of each $\min(w_{2i-1}, w_{2i})$ equals $1/3$. □

Fact 9.3 *The probability that any specific perfect matching* (and in particular the one of the second type) *has weight* less than*, say, $0.31 \cdot n$ is greater than $\frac{0.6^n}{2} = \exp(\Omega(n)) \cdot 2^{-n}$.*

[2] This parenthetical comment is based on discussions with Madhu Sudan (during March 1998).

Proof Sketch: This follows by observing that

$$\begin{aligned}
&\mathsf{Pr}\left[\sum_{i=1}^{n} w_i < 0.31 \cdot n\right] \\
&> \mathsf{Pr}[\forall i\,(w_i \leq 0.6)] \cdot \left(1 - \mathsf{Pr}\left[\sum_{i=1}^{n} w_i \geq 0.31 \cdot n \;\middle|\; \forall i\,(w_i \leq 0.6)\right]\right) \\
&> 0.6^n \cdot \frac{1}{2}
\end{aligned}$$

where the last inequality uses $\mathsf{E}[w_i \,|\, w_i \leq 0.6] = 0.3$. ■

Combining Facts 9.2 and 9.3, we conclude that, with probability $\exp(\Omega(n)) \cdot \frac{1}{2^n+1}$, the single (second type) matching has weight *less than* $0.31 \cdot n$ and every perfect matching of the first type has weight *at least* $0.32 \cdot n$. In this case, the single (second type) matching is of minimum-weight among all $2^n + 1$ perfect matchings, and the claim follows.

9.1.3 Testing whether polynomials are identical or, on the discrete charm of polynomials

The problem considered here is to determine whether two multi-variant polynomials are identical. We assume that one is given an oracle for the evaluation of each of the polynomials. We further assume that the polynomials are defined over a sufficiently large finite field, denoted F. Finally, let n denote the number of variables in these polynomials.

Algorithm: Given n and black-box access to $p, q : \mathrm{F}^n \to \mathrm{F}$, uniformly select $r_1, ..., r_n \in \mathrm{F}$, and accept if and only if $p(r_1, ..., r_n) = q(r_1, ..., r_n)$.

Analysis: Clearly, if $p \equiv q$ then the algorithm always accepts. The following lemma implies that if p and q are different polynomials, each of total degree at most d, then the algorithm accepts with probability at most $d/|\mathrm{F}|$.

Lemma 9.4 *Let $p : \mathrm{F}^n \to \mathrm{F}$ be a non-zero polynomial of total degree d. Then*

$$\mathsf{Pr}_{r_1,...,r_n}[p(r_1, ..., r_n) = 0] \;\leq\; \frac{d}{|\mathrm{F}|}$$

Proof: The lemma is proven by induction on n. The base case of $n = 1$ follows immediately by the Fundamental Theorem of Algebra (i.e., the number of distinct roots of a degree d univariant polynomial is at most d). In the induction step, we write p as a polynomial in its first variable. That is,

$$p(x_1, x_2, ..., x_n) = \sum_{i=0}^{d} p_i(x_2, ..., x_n) \cdot x_1^i$$

where p_i is a polynomial of total degree at most $d-i$. Let t be the biggest integer i for which p_i is not identically zero. (We dismiss the case $t = 0$.) Then, using the induction hypothesis, we have

$$\begin{aligned}\mathsf{Pr}_{r_1,r_2,...,r_n}[p(r_1,r_2,...,r_n)=0] &\leq \mathsf{Pr}_{r_2,...,r_n}[p_t(r_2,...,r_n)=0] \\ &\quad + \mathsf{Pr}_{r_1,r_2,...,r_n}[p(r_1,r_2,...,r_n)=0 \,|\, p_t(r_2,...,r_n)\neq 0] \\ &\leq \frac{d-t}{|\mathrm{F}|}+\frac{t}{|\mathrm{F}|}\end{aligned}$$

where the second term is bounded by fixing any sequence $r_2,...,r_n$ for which $p_t(r_2,....,r_n) \neq 0$ and considering the univariant polynomial $p'(x) \stackrel{\text{def}}{=} p(x,r_2,...,r_n)$, which by hypothesis is a non-zero polynomial of degree t. ■

Comment: The lesson is that whenever the situation is such that *almost* any choice will do, taking a random choice yields an algorithm with a rigorous performance guarantee. In a sense any randomized algorithm is based on this paradigm, except that here the space of choices seems more straightforward than in any other case. That is, most randomized algorithms are based on introducing a sample space that is not obvious from the problem at hand; whereas here the sample space is the obvious one.

9.1.4 Randomized Rounding applied to MaxSAT or, on being fractionally pregnant

We slightly deviate from the above style of exposition by considering a general methodology for approximating combinatorial optimization problems. The methodology, which relies on the fact that linear programming is solveable in polynomial-time, consists of two steps. First, one presents a linear programming relaxation of an integer program (corresponding to a combinatorial problem). Next, one derives from a solution to the linear program (LP) a solution to the integer program (IP). This is done by using the former (LP) solution in order to determine a probability distribution over integer solutions (i.e., solution to the IP), and picking a solution according to this distribution. We exemplify this methodology by applying it to Max-SAT. Specifically, we consider the task of approximating the maximum number of clauses that can be simultaneously satisfied in a given CNF formula.

Let $\varphi = \bigwedge_{j=1}^{m} C_j$ be a CNF formula, where $C_j = (\bigvee_{i\in S_j^+} x_i) \vee (\bigvee_{i\in S_j^-} \neg x_i)$ with $S_j^+, S_j^- \subseteq [n] \stackrel{\text{def}}{=} \{1,...,n\}$. Abusing notation, we may express Max-SAT as an integer optimization problem in which the task is to maximize $\sum_{j=1}^{m} y_j$ subject to

$$x_i, y_j \in \{0,1\} \qquad (\forall i,j) \tag{9.3}$$

$$\sum_{i\in S_j^+} x_i \;+\; \sum_{i\in S_j^-} (1-x_i) \;\geq\; y_j \qquad (\forall j) \tag{9.4}$$

In the Linear Programming (LP) relaxation, one replaces Eq. (9.3) by

$$0 \leq x_i, y_j \leq 1 \quad (\forall i,j) \tag{9.5}$$

Clearly, the value of the LP is lower bounded by the value of the integer program. Given an (optimal) solution, $\hat{x}_i, \hat{y}_j$, to the LP, we randomly derive a solution to the original integer formulation. It will be shown that the expected value of the integer solution is at least $1 - e^{-1}$ times the value of the LP (and hence at least a $1 - e^{-1}$ fraction of the optimum of the integer problem). Specifically, we set $x_i = 1$ with probability $\hat{x}_i$ (and $x_i = 0$ otherwise).

Analysis: Suppose that clause C_j has c_j literals. Below, we will show that the probability that C_j is satisfied by the integer assignment (generated by above randomized rounding of the above LP solution) is at least

$$\left(1 - \left(1 - \frac{1}{c_j}\right)^{c_j}\right) \cdot \hat{y}_j \;\geq\; \left(1 - e^{-1}\right) \cdot \hat{y}_j$$

and so the expected number of satisfied clauses is at least $(1-e^{-1}) \cdot \sum_j \hat{y}_j$ (as stated above). The above is proven by noting that the probability of the complementary event (i.e., C_j is not satisfied) is

$$\left(\prod_{i \in S_j^+} (1 - \hat{x}_i)\right) \cdot \left(\prod_{i \in S_j^-} \hat{x}_i\right) \tag{9.6}$$

where, by Eq. (9.4), $\sum_{i \in S_j^+}(1 - \hat{x}_i) + \sum_{i \in S_j^-} \hat{x}_i \leq (c_j - \hat{y}_j)$. Eq. (9.6) is maximized when $1 - \hat{x}_i = (c_j - \hat{y}_j)/c_j$ for all $i \in S_j^+$, and $\hat{x}_i = (c_j - \hat{y}_j)/c_j$ for all $i \in S_j^-$. Thus, Eq. (9.6) is bounded above by $\left(1 - \frac{\hat{y}_j}{c_j}\right)^{c_j}$, and the above claim follows.

Comments: Combining the above algorithm with the naive algorithm that uniformly selects a truth assignment, one derives a randomized algorithm of a 3/4-approximation factor. The key observation is that the performance of the LP-based algorithm improves as the clause sizes decrease, whereas the performance of the naive algorithm improves when the sizes increase. In a different vein, we mention that the randomized rounding paradigm has been extended also to *semidefinite* (rather than linear programming) relaxations of combinatorial problems. In fact, improved approximation ratios for various versions of MaxSAT were obtained that way (cf., [53, 70]).

9.1.5 Primality Testing or, on hiding information from an algorithm

The problem considered here is to decide whether a given number is a prime. The only Number Theoretic facts that we use are:

Fact 9.5 *For every prime $p > 2$, each quadratic residue mod p has exactly two square roots mod p (and they sum-up to p).*

Fact 9.6 *For every (odd and non-integer-power) composite number N, each quadratic residue mod N has at least four square roots mod N.*

Our algorithm uses as a black-box an algorithm, denoted R, that given a prime p and a quadratic residue mod p, returns the smallest among the two square roots. There is no guarantee as to what is the output in case the input is not of the above form (and in particular in case p is not a prime).

Algorithm: On input a natural number $N > 2$ do

1. If N is either even or an integer-power then `reject`.
2. Uniformly select $r \in \{1, ..., N-1\}$, and set $s \leftarrow r^2 \bmod N$.
3. Let $r' \leftarrow R(N, s)$. If $r' \equiv \pm r \pmod{N}$ then `accept` else `reject`.

Analysis: By Fact 9.5, on input a prime number N, the above algorithm always accepts (since in this case $R(N, r^2 \bmod N) = \pm r$ for any $r \in \{1, ..., N-1\}$). On the other hand, suppose that N is an odd composite that is not an integer-power. Then, by Fact 9.6, each quadratic residue s has at least four square roots, and each is equally likely to be chosen at Step 2 (becuase s yields no information on the specific r). Thus, for every such s, the probability that $\pm R(N, s)$ has been chosen in Step 2 is at most $2/4$. It follows that, on input a composite number, the algorithm rejects with probability at least $1/2$.

Comment: The above analysis presupposes that the algorithm R is always correct when fed with a pair (p, s), where p is prime and s a quadratic residue mod p. In case R has error probability $\varepsilon < 1/2$, our algorithm still distinguishes primes from composites (since on the former it accepts with probability at least $1 - \varepsilon > 1/2$). We note that efficient randomized algorithms for extracting square roots modulo a prime are known (cf., [14, 83]). Thus, the above establishes that primality can be decided in probabilistic polynomial-time (alas, with two-sided error).

Later comment: We mention that a deterministic polynomial-time algorithm was found for this problem a few years ago by Agrawal, Kayal, and Saxena [1].

9.1.6 Testing Graph Connectivity via a random walk or, the accidental tourist sees it all

The problem considered here is to decide whether a given graph is connected. The aim is to devise an algorithm that does so while using little space (i.e., essentially, as little as needed for storing the identity of a single vertex). This task can be reduced (in small space) to testing connectivity between any given pair of vertices. Thus, we focus on the task of determining whether or not two given vertices are connected in a given graph.

Algorithm: On input a graph $G = (V, E)$ and two vertices, s and t, we take a *random walk* of length $O(|V| \cdot |E|)$, starting at vertex s, and test at each step whether or not vertex t is encountered. By a random walk we mean that, at each step, we uniformly select one of the edges incident at the current vertex and traverse this edge to the other endpoint.

Analysis: We will show that if s is connected to t in the graph G then, with probability at least $1/2$, vertex t is encountered in a random walk starting at s. In the following, we consider the connected component of vertex s, denoted $G' = (V', E')$. For any edge, (u, v) (in E'), we let $T_{u,v}$ be a random variable representing the number of steps taken in a random walk starting at u until v is first encountered. It can be shown that $\mathrm{E}[T_{u,v}] \leq 2|E'|$.[3] Also, letting cover(G') be the expected number of steps in a random walk starting at s and ending when the last of the vertices of V' is encountered, and C be any directed cycle that visits all vertices in G', we have

$$\begin{aligned} \mathrm{cover}(G') &\leq \sum_{(u,v)\in C} \mathrm{E}[T_{u,v}] \\ &\leq |C| \cdot 2|E'| \end{aligned}$$

Letting C be a traversal of some spanning tree of G', we conclude that $\mathrm{cover}(G') < 4 \cdot |E'| \cdot |V'|$. Thus, with probability at least $1/2$, a random walk of length $8 \cdot |E'| \cdot |V'|$ starting at s visits all vertices of G'.

Comment: We mention that a deterministic log-space algorithm was found for this problem a few years ago by Reingold [90]. The interested reader may also find a description of it in [44, Sec. 5.2.4].

9.1.7 Finding minimum cuts in graphs or, random is better than arbitrary

The problem considered here is to find the minimum cut in a graph. The randomized algorithm that follows is simpler than the traditional flow-based algorithms, and lends itself to parallel implementation (omitted here).

Algorithm: On input a graph $G = (V, E)$, with $n = |V|$, the algorithm makes $n - 2$ random edge *contraction* steps: In each step one selects *uniformly* an edge of the current multi-graph and contracts the two endpoints into one vertex, allowing parallel edges but dropping self-loops that may be created. That is, if (u, v) is the contracted edge of the current graph G' then we replace vertices u and v by a new vertex x, and replace edges of the form (w, v) (resp., (w, u)), where $w \notin \{u, v\}$, by a similar number of edges (w, x). When these $n - 2$ contraction steps are completed, we are left with a multi-graph on two vertices, and just output the number of parallel edges.

[3] A hand-waving argument follows: Consider a very long walk, starting at u, and returning to u many times. Note that each directed edge appears on this walk for about the same number of times. Partition the walks into segments so that each segment ends with a move from vertex v to vertex u. Then, the number of segments is about a $1/2|E'|$ fraction of the length of the walk, and so the average length of a segment is $2|E'|$. Note that each segment constitutes a walk starting at u and passing through v (possiblly several times) before returning to u. Thus, the average length of segments in the big walk upper bounds the expected length of random walks from u to v.

Analysis: Suppose that G has a minimum cut $C \subset E$. Then, the probability that no edge of C is contracted in the first step is $\frac{|E|-|C|}{|E|} \geq 1 - \frac{2}{n}$ (where $|C| \leq 2|E|/n$ because the minimum cut cannot be bigger than the average degree). The question is what happens in subsequent steps. A key observation is that $|C|$ is a lower bound on the average degree of any multi-graph obtained from G by any sequence of edge contractions. Thus, the probability that the $(n-2)$-step contraction process leaves C intact is at least

$$\prod_{i=1}^{n-2}\left(1 - \frac{2}{n-(i-1)}\right) \;=\; \prod_{i=1}^{n-2}\frac{n-1-i}{n+1-i} \;=\; \frac{2}{n\cdot(n-1)}$$

Thus, repeating the above algorithm for a quadratic number of times we obtain the minimum cut, with probability at least, say, 2/3. We comment that it follows that the number of minimum cuts is at most $(n-1)n/2$ (becuase each such cut is generated by the above algorithm with probability at least $2/(n-1)n$).

Comment: Observe that if the random choices in the above algorithm are replaced by arbitrary choices then the output gives little indication towards the minimum cut in G. That is, an algorithm that makes $n-2$ arbitrary edge-contraction steps provides no useful information, whereas the above algorithm that picks these steps at random is useful. In general, making random choices may be better than making arbitrary choices. This lesson is important because many algorithms are presented in a non-fully specified manner, allowing some choices to be made arbitrarily (in which case these choices are typically made in a way most convenient for implementation). It is important to bear in mind that, in some cases, replacing an arbitrary choice by a random one may yield improved performance.

9.2 Randomness in Complexity Theory

In this section we demonstrate the power of randomized reductions (rather than randomized algorithms discussed in the previous section). Again, we focus on simple examples, and avoid the central role of randomness in the context of proof systems. For a survey of probabilistic proof systems, the interested reader is referred to [41, Chap. 2].

9.2.1 Reducing (Approximate) Counting to Deciding or, the Random Sieve

We consider the class $\#\mathcal{P}$ of functions that count the number of NP-witnesses (w.r.t an NP-relation). That is, $f \in \#\mathcal{P}$ if for some NP-relation, R, it holds that $f(x) = |\{y : (x,y) \in R\}|$, for every $x \in \{0,1\}^*$. We will show that such f can be approximated in probabilistic polynomial-time given oracle to an NP-complete set. The (randomized Cook) reduction uses any efficient family of Universal$_2$ Hash functions[4], as well as the following lemma.

[4] A family of functions mapping $\{0,1\}^m$ to $\{0,1\}^k$ is called Universal$_2$ if for a uniformly selected h in the family, the random variables $\{h(e)\}_{e\in\{0,1\}^m}$ are pairwise independent and uniformly distributed over $\{0,1\}^k$. An efficient family is required to have algorithms for selecting and evaluating functions. A popular example is the family of all affine transformations from $\{0,1\}^m$ to $\{0,1\}^k$.

Lemma 9.7 (Leftover Hash Lemma [96, 23, 61]):[5] *Let $H_{m,k}$ be a family of* Universal$_2$ *Hash functions mapping $\{0,1\}^m$ to $\{0,1\}^k$, and let $\varepsilon > 0$. Let $S \subseteq \{0,1\}^m$ be arbitrary provided that $|S| \geq \varepsilon^{-3} \cdot 2^k$. Then, for all but at most an ε fraction of the h's in $H_{m,k}$, it holds that*

$$|\{e \in S : h(e)=0^k\}| \;=\; (1 \pm \varepsilon) \cdot \frac{|S|}{2^k}$$

Proof: For a uniformly selected $h \in H_{m,k}$, the random variables $\{h(e)\}_{e \in S}$ are pairwise independent and uniformly distributed over $\{0,1\}^k$. On top of these $h(e)$'s, we define 0-1 random variables, denoted ζ_e's, so that $\zeta_e = 1$ if $h(e) = 0^k$. Then $\mathsf{E}[\zeta_e] = 2^{-k}$ and we need to show that the sum $\sum_{e \in S} \zeta_e$ is concentrated around $|S|/2^k$. Using Chebyshev's Inequality and the fact that the ζ_e's are pairwise independent, we get

$$\begin{aligned} \mathsf{Pr}\left[\left|\sum_{e \in S} \zeta_e - \frac{|S|}{2^k}\right| > \frac{\varepsilon \cdot |S|}{2^k}\right] &< \frac{\mathsf{Var}[\sum_{e \in S} \zeta_e]}{(\varepsilon |S|/2^k)^2} \\ &< \frac{|S|/2^k}{\varepsilon^2 \cdot (|S|/2^k)^2} \;\leq\; \varepsilon \end{aligned}$$

(Pairwise independence is used in deriving $\mathsf{Var}[\sum_{e \in S} \zeta_e] = \sum_{e \in S} \mathsf{Var}[\zeta_e] < |S| \cdot 2^{-k}$.) ■

Reduction: On input $x \in \{0,1\}^n$, the probabilistic polynomial-time oracle machine (for approximating f) sets m to be the length of NP-witness w.r.t the guaranteed R. For every $k = 0, 1, ..., m+2$ it performs the following experiment n times.

1. Uniformly select $h \in H_{m,k}$, and construct (via Cook's reduction) a CNF formula φ so that φ is satisfiable if and only if there exists a string $y \in \{0,1\}^m$ so that $(x,y) \in R$ and $h(y) = 0^k$.
2. Query the oracle whether φ is satisfiable.

Once all these experiments are completed, the machine determines the smallest non-negative integer k (possibly zero) so that the oracle has answered NO at least $n/2$ times, and outputs 2^k.

Analysis: We analyze the performance of the above machine when it is given oracle access to SAT. Clearly, if $S_x \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$ has cardinality N then the probability that the machine outputs a number $k \geq L \stackrel{\text{def}}{=} \lceil \log_2(4N) \rceil$ is exponentially vanishing (because the probability that a uniformly selected $h \in H_{m,L}$ maps some element of S_x to 0^L is at most $1/4$, and so in each iteration with value of $k \geq L$, with probability at least $3/4$, the oracle says NO). On the other hand, using the above lemma, if $N \stackrel{\text{def}}{=} |S_x| \geq 2^{k+2}$ then for a uniformly selected $h \in H_{m,k}$ with probability at least $3/4$ there exists $y \in S_x$ so that

[5] A stronger statement of the lemma, supported by essentially the same proof, refers to an arbitrary random variable X over $\{0,1\}^m$ satisfying $\mathsf{Pr}[X = x] \leq \varepsilon^3 \cdot 2^{-k}$, for every x. The lemma was discovered independently in [23, 61], yet it is an extension of the ideas underlying [96]. The lemma's name was coined in [62].

$h(y) = 0^k$. Thus, with overwhelmingly high probability, the output of the oracle machine is at least $\log_2(N/4)$. We conclude that approximating f up-to a factor of 4 is reducible in probabilistic polynomial-time to $\mathcal{NP}$. Higher accuracy – that is, approximation factor of $1 + \frac{1}{p(n)}$, for any fixed positive polynomial p – can be obtained by considering the "direct product function" $F_p(x) \stackrel{\text{def}}{=} (f(x))^{p(|x|)}$ that counts the number of NP-witnesses w.r.t the NP-relation R_p defined by

$$R_p \stackrel{\text{def}}{=} \{(x, y_1, ..., y_{p(|x|)}) : \forall i \ (x, y_i) \in R\}$$

A related reduction may be used to reduced SAT (or even "approximating $\#\mathcal{P}$") to unique-SAT. By the latter, we mean the promise problem in which the YES-instances are CNF formula having a unique satisfying assignment, and the NO-instances are CNF formula having no satisfying assignment. All that is needed is to notice that in the above reduction, for $k = (\log_2 N) \pm 2$, the reduction produces CNF formula that are typically (i.e., w.p. at least 3/4) either not satisfiable or have few (say up-to 8) satisfying assignments. Thus, we augment Step 1 as follows. Having produced φ, as above, we produce 8 new formulae, $\psi_1, ..., \psi_8$, so that ψ_i asserts that φ has at least i different satisfying assignments (e.g., $\psi_i(y_1, ..., y_i) = \bigwedge_j \varphi(y_j) \wedge \bigwedge_{1 \leq j < j' \leq i}(y_j < y_{j'})$). We refer each of these ψ_i to the oracle and use YES as answer if the oracle has answered YES on any of the ψ_i (as this may happen only if φ is indeed satisfiable). Thus, whenever φ has few satisfying assignments, YES will be returned.

9.2.2 Two-sided error versus one-sided error

We consider the extension of the classes $\mathcal{RP}$ and $\mathcal{BPP}$ to promise problems and show that $\mathcal{BPP} = \mathcal{RP}^{\mathcal{RP}}$ (in the extended sense). It is evident that $\mathcal{RP}^{\mathcal{RP}} \subseteq \mathcal{BPP}^{\mathcal{BPP}} = \mathcal{BPP}$ (where the last equality utilizes standard "error reduction"). So we focus on the other direction, considering an arbitrary BPP-problem with a characteristic function χ (which may be only partially defined over $\{0,1\}^*$). Let R be an NP-relation and p be a polynomial, such that for every x on which χ is defined it holds that

$$|\{y \in \{0,1\}^{p(|x|)} : R(x,y) \neq \chi(x)\}| \ < \ \frac{2^{p(|x|)}}{3p(|x|)}$$

where $R(x,y) = 1$ if $(x,y) \in R$ and $R(x,y) = 0$ otherwise. We show a randomized one-sided error (Karp) reduction of χ to (the promise problem extension of) co$\mathcal{RP}$.

Reduction: On input $x \in \{0,1\}^n$, the randomized polynomial-time mapping uniformly selects $s_1, ..., s_m \in \{0,1\}^m$, and outputs the pair $(x, \overline{s})$, where $m = p(|x|)$ and $\overline{s} = (s_1, ..., s_m)$.

We define the following co$\mathcal{RP}$ promise problem, denoted Π. The YES-instances, denoted Π_{yes}, are pairs $(x, \overline{s})$ so that for every $r \in \{0,1\}^m$ there exists an i so that $R(x, r \oplus s_i) = 1$. The NO-instances, denoted Π_{no}, are pairs $(x, \overline{s})$ so that for at least half of the possible $r \in \{0,1\}^m$, it holds that $R(x, r \oplus s_i) = 0$ for every i. Clearly, Π is indeed a co$\mathcal{RP}$ promise problem (via an algorithm that uniformly selects r, and computes $R(x, r \oplus s_i)$ for all i's).

Analysis: We claim that the above randomized mapping reduces χ to Π. Suppose first that $\chi(x) = 0$. Then, for every possible choice of $s_1, ..., s_m \in \{0,1\}^m$, the fraction of r's for which $R(x, r \oplus s_i) = 1$ holds for some i is at most $m \cdot \frac{1}{3m} = \frac{1}{3}$. Thus, the reduction always maps such an x to a NO-instance (i.e., an element of Π_{no}). On the other hand, we will show shortly that in case $\chi(x) = 1$, with probability at least $1/2$ the reduction maps x to a YES-instance. Thus, the above reduction has one-sided error and indeed reduces χ to Π (which, as observed above, is in co$\mathcal{RP}$). It is left to analyze the probability that the reduction fails in case $\chi(x) = 1$. That is,

$$\begin{aligned}
\mathsf{Pr}_{\overline{s}}[(x,\overline{s}) \notin \Pi_{\text{yes}}] &= \mathsf{Pr}_{s_1,...,s_m}[\exists r \in \{0,1\}^m \text{ s.t. } (\forall i)\ R(x, r \oplus s_i) = 0] \\
&\leq \sum_{r \in \{0,1\}^m} \mathsf{Pr}_{s_1,...,s_m}[(\forall i)\ R(x, r \oplus s_i) = 0] \\
&\leq 2^m \cdot \left(\frac{1}{3m}\right)^m \ll \frac{1}{2}
\end{aligned}$$

Comment: The traditional presentation uses the above reduction to show that $\mathcal{BPP}$ is in the Polynomial-Time Hierarchy. One defines the polynomial-time predicate $\varphi(x, \overline{s}, r) \stackrel{\text{def}}{=} \bigvee_{i=1}^m (R(x, s_i \oplus r) = 1)$, and observes that

$$\begin{aligned}
\chi(x) = 1 &\Rightarrow \exists \overline{s}\, \forall r\ \varphi(x, \overline{s}, r) \\
\chi(x) = 0 &\Rightarrow \forall \overline{s}\, \exists r\ \neg\varphi(x, \overline{s}, r)
\end{aligned}$$

9.2.3 The permanent: Worst-Case versus Average Case or, the self-correction paradigm

We consider the problem of computing the permanent of a matrix.[6] This problem is known to be $\#\mathcal{P}$-complete even in case the matrix has only 0-1 entries. Here we consider the problem of computing the permanent over sufficiently large finite fields (i.e., the field size is larger than the dimension). We show that the (worst-case) problem can be reduced to solving the problem on random (or typical) instances.

Reduction: On input an n-by-n matrix, M, over F (s.t., $|F| > n + 1$), the probabilistic polynomial-time oracle machine (i.e., the reduction) proceeds as follows.

1. Uniformly select an n-by-n matrix, R, over F.
2. For $i = 1, ..., n + 1$, obtain from the oracle the value, denoted v_i, of the permanent of the matrix $M + iR$.
3. Obtain by interpolation, the value of the degree n univariant polynomial, p, satisfying $p(i) = v_i$ (for $i = 1, ..., n + 1$).
4. Output $p(0)$.

The key observation, underlying the above reduction, is that, for fixed M and R, the permanent of $M + iR$ is a degree n polynomial in the variable i.

[6] The permanent of an n-by-n matrix $A = (a_{i,j})$ is the sum, taken over all permutations π of $[n]$, of the products $\prod_{i=1}^n a_{i,\pi(i)}$.

Analysis: We consider the performance of the above reduction assuming it is given access to an oracle that answers correctly on all but at most an $1/3(n+1)$ fraction of the instances. We will show that in such a case, on any input, the reduction answers correctly with probability at least $2/3$. Observe that, for each fixed M and $i \neq 0$, the matrix $M+iR$ is uniformly distributed over the instance space. Thus, the probability that the oracle returns an incorrect answer on any of the $n+1$ queries is at most $1/3$. But otherwise, having the permanent of $M+iR$ for every $i = 1, .., n+1$, we obtain the permanent of the formal matrix $M+xR$ (which is a polynomial of degree n in $x \in \mathrm{F}$), and thus the permanent of M (when substituting $x = 0$).

Comments: As seen above, the reduction of a problem to random instances of itself allows to reduce its "worst" instances to its average (or typical) cases, and thus means that the problem does not really have "worst" (or "pathological") instances: The problem's complexity, in case the problem is hard, must stem from typical (or random) instances. Viewed from the other side (i.e., of feasibility), such a reduction allows to *self-correct* a (possibly efficient) procedure that is correct on a large majority of the instances, and obtain a randomized procedure that is correct on every instance. Thus, as any reduction, a reduction to random instances is open to interpretation: For example, Ajtai's reduction of approximating shortest vectors in integer lattices to such random instances [2], is commonly viewed as a demonstration of average-case hardness based on worst-case hardness, but it may be also viewed as a self-corrector for (possibly efficient) programs that find short vectors in a certain class of integer lattices.

9.3 Randomness in Distributed Computing

As much as randomness is a powerful tool in the design of algorithms and reductions, its power in the distributed context is even more striking. In particular, randomized distributed protocols are known to beat some impossibility results and lower bound that refer to deterministic protocols. Various examples are given in [29, 77, 13, 72, 83].

As a warm-up consider the problem of electing a leader among a set of n *identical* processes. Clearly, there is no deterministic procedure to elect such a leader (even when all processes are guaranteed to be non-faulty), because there is no way to "*deterministically* break the symmetry" among the processors. However, a simple randomized procedure will do the job: Let each processor toss, independently of all other processors, a coin with bias $1/n$ towards 1, and announce its coin-flip to all processors. If a single processor sends 1 then it is elected leader, otherwise the process is repeated. In general, randomness can be used to "break symmetry" in a variety of distributed settings. Other uses of randomness in such settings include avoiding "pathological" configurations (see Section 9.3.2), and making the actions of non-faulty processors unpredictable to malicious ones (i.e., Byzantine faults; see Section 9.3.3). We start with a much simpler problem.

9.3.1 Testing String Equality or, randomized fingerprints

The problem considered here is to decide whether or not two strings, each held by a different party, are identical. The aim is to devise a protocol for this problem using low communication complexity. We present three such protocols.

Protocol 1: Party A holds $x \in \{0,1\}^n$, whereas party B holds $y \in \{0,1\}^n$. Here we view x and y as non-negative integers in $\{0,1,...,2^n-1\}$. In the protocol, party A uniformly selects $i \in \{1,...,n\}$, finds the i^{th} prime, denoted p_i, and sends the pair $(i, x \bmod p_i)$ to B. Party B recovers p_i and accepts if and only if $y \bmod p_i$ equals the value $x \bmod p_i$ (received from A).

Clearly, if $x = y$ then B always accepts. On the other hand, using the Chinese Reminder Theorem, we know that if $x \neq y$, then $x \neq y \pmod{p_i}$ for at least $n/2$ of the p_i's (since otherwise $x \equiv y \pmod{\prod_{i \in I} p_i}$, for $|I| \geq n/2$, and $x = y$ follows (because $x, y < 2^n < \prod_{i \in I} p_i$)). Thus, B will reject with probability at least $1/2$. The number of bits sent is $\log_2 n + \log_2 p_n = O(\log n)$.

Protocol 2: Again, party A holds $x \in \{0,1\}^n$, whereas party B holds $y \in \{0,1\}^n$. Here we use a small-bias probability space $S \subset \{0,1\}^n$, with bias $1/3$ and $|S| = \text{poly}(n)$ (see Section 3.3). By definition, for every non-zero string $z \in \{0,1\}^n$, with probability at least $1/3$ a uniformly chosen $r \in S$ has inner product mod 2 with z equal to 1. In the protocol, party A uniformly selects $r \in S$, computes the inner product mod 2 of x and r, and sends the result along with the index of r (in S) to B. Party B retrieves r, computes the inner product mod 2 of y and r, and accepts if it matches the bit received.

Clearly, if $x = y$ then B always accepts. On the other hand, by the above, if $x \neq y$ then the inner products of x and y with a uniformly chosen $r \in S$ differ with probability at least $1/3$ (hint: consider $z = x \oplus y$). The number of bits sent is $1 + \log_2 |S| = O(\log n)$.

Protocol 3: The inputs are as above, but here we use a different tool: An error-correcting code, denoted $E : \{0,1\}^n \to \{0,1\}^m$, with $m = O(n)$ and distance $\Omega(n)$ (cf., Section 2.3). In the protocol, party A computes the codeword $E(x)$, uniformly selects $i \in \{1,...,m\}$, and sends i along with the i^{th} bit of $E(x)$ to Party B. The latter computes the codeword $E(y)$ and accepts if its i^{th} bit matches the bit received.

Clearly, if $x = y$ then B always accepts. On the other hand, if $x \neq y$ then $E(x)$ and $E(y)$ differ on a constant fraction of the bit positions, and so B will reject with constant probability. The number of bits sent is $1 + \log_2 m = O(1) + \log_2 n$.

9.3.2 Routing in networks or, avoiding pathological configurations

The problem considered here is to allow parallel routing of messages in a network in which processors have relatively few immediate neighbors (i.e., processors connected to them by a direct link). In many such networks, *routing to random destinations* can be done quite efficiently (i.e., fast even assuming that each processor can only deliver a single message at a time, and without coordination among the processors). Off course, we are interested in routing messages to "non-random" destinations; that is, to destinations that are imposed upon us by some high-level application. Still the above fact (regarding routing to random

destinations) becomes relevant, via the following two phase *randomized routing* strategy: Suppose that processor i wishes to deliver a message to processor d_i, where the d_i's consist of an arbitrary a permutation of the processor names $[n] \stackrel{\text{def}}{=} \{1, ..., n\}$. Then, processor i selects a random intermediate processor, $r_i \in [n]$, and sends its message to processor r_i with a request to forward it to processor d_i. (The r_i's are not likely to be distinct!) Thus, the routing is in two phases:

1. The message of processor i, denoted m_i, is delivered to r_i.
2. Message m_i is delivered from r_i to d_i.

By our hypothesis, Phase 1 can be completed fast with high probability. It is appealing to say that, by symmetry, the same should hold also for Phase 2. This is not known to be generically true, but has been proved to be so for a wide class of networks (cf., [74, Sec. 3.4]). Specifically, if one changes the model a little, allowing and measuring edge congestion, then bounds on congestion in Phase 1 apply also to Phase 2.

9.3.3 Byzantine Agreement or, take actions the adversary cannot predict

The problem considered here is to allow non-faulty processors to agree on a common value, in presence of Byzantine (malicious) faulty processors. Specifically, it is required that (1) the non-faulty processors must terminate with the same output value, and (2) in case their input values are the same this should also be their output value. We may consider, without loss of generality, the problem of agreeing on a Boolean value. The primary parameters are the total number of processors, denoted n, and a bound on the number of faulty processors, t. We assume a synchronous model of point-to-point communication.

Protocol: We use auxiliary (threshold) parameters L, H, D so that $L > \frac{n}{2} + t$, $H \geq L + t$ and $H + t \leq D \leq n - t$ (which is feasible for $t < n/8$). The protocol utilizes a *global coin* (which may be implemented in various ways). It is postulated that, for each flipping of this coin, each of the two possible outcomes occurs with probability at least $p > 0$ ($p = 0.1$ will do, whereas $p = 0.5$ corresponds to an unbiased coin).

Following is the program to processor $i \in [n] \stackrel{\text{def}}{=} \{1, ..., n\}$. On input $b_i \in \{0, 1\}$, the processor sets its (initial) vote, denoted $\texttt{vote}_i$, to b_i. The processor repeats the following steps $r + 1$ times, where r is the iteration in which it decides (see below):

1. Send $\texttt{vote}_i$ to each processor.
2. Receive votes from all processors, including itself. (In case no message is received from processor j, use the value last received from it, and if no value was ever received use value 0.) Let $\texttt{cnt}_i$ denote the number of votes in favor of 1. If $\texttt{cnt}_i > n/2$ set $\texttt{maj}_i = 1$ and $\texttt{tally}_i = \texttt{cnt}_i$, otherwise set $\texttt{maj}_i = 0$ and $\texttt{tally}_i = n - \texttt{cnt}_i$.
3. Let $C \in \{L, H\}$ be the value of the global coin, for the current round (in each round the global coin is flipped anew).

4. If $\texttt{tally}_i \geq C$ then set $\texttt{vote}_i = \texttt{maj}_i$ else set $\texttt{vote}_i = 0$.
5. If $\texttt{tally}_i \geq D$ then *decide* $\texttt{vote}_i$, and proceed for a single additional iteration (skipping this step in the next iteration).

   (Actually, as shown below, if the processor were to decide again in the next iteration its decision would have been identical.)

Analysis: Let G denote the set of non-faulty (or good) processors. The following observation regarding members of G is extensively used: In each iteration, $|\texttt{cnt}_i - \texttt{cnt}_j| \leq t$, for every $i, j \in G$. Thus, if $\texttt{tally}_i \geq L > n/2 + t$ for some $i \in G$ then $\texttt{maj}_j = \texttt{maj}_i$ for all $j \in G$. Similarly, if $\texttt{tally}_i \geq D$ (resp., $\texttt{tally}_i \geq H$) for some $i \in G$ then $\texttt{tally}_j \geq H$ (resp., $\texttt{tally}_j \geq L$) for all $j \in G$. Using these facts it follows that

1. *If all good processors enter some round with identical votes then they all decide by the end of the current round, and their decision equals this vote.* This follows since (at this round) this identical vote would have support of at least $|G| \geq n - t \geq D$. (As a special case, we conclude that the second requirement of Byzantine Agreement holds.)
2. *If at some round a good processor decides v then by the end of the next round all good processors decide v.* Suppose that $i \in G$ decides v in the current round. Then, $\texttt{tally}_i \geq D$, and for each $j \in G$ it follows that $\texttt{tally}_j \geq H$ and so at Step 4 $\texttt{vote}_j = \texttt{maj}_j = v$. Using the previous fact, the current one follows. (As a special case, we conclude that the first requirement of Byzantine Agreement holds.)
3. *If at some round $\texttt{tally}_i \geq H$ holds for some $i \in G$ then with constant probability all good processors enter the next round with vote equal to $\texttt{maj}_i$.* This follows since with constant probability the outcome of the global coin is L, in which case for every $j \in G$, $\texttt{tally}_j \geq L = C$ and so at Step 4 $\texttt{vote}_j = \texttt{maj}_j = \texttt{maj}_i$.
4. *If at some round $\texttt{tally}_i < H$ holds for all $i \in G$ then with constant probability all good processors enter the next round with vote 0.* This follows since with constant probability the outcome of the global coin is H.

Thus, the above protocol terminates in *constant expected number of rounds*, and the output always satisfies the agreement requirements. This remain valid even if we use a global coin the outcome of which may be viewed differently by different processors, as long as for each of the two possible values, with probability at least $p > 0$, all non-faulty processors view the outcome as equal to that value. We comment that such a global coin can be easily implemented in case $t = O(\sqrt{n})$, by letting each processor toss a local coin, announce the outcome, and view the outcome of the global coin to be the majority vote it has received (which, with constant probability, will be identical at all good processors). We note that $t + 1$ is a lower bound on the number of rounds in any correct *deterministic* protocol. Furthermore, the above protocol can be adapted to the asynchronous model, whereas there exist no correct *deterministic* protocol for the latter model (even for $t = 1$).

Notes

We briefly list the credits for the various randomized computations described in this section.

Algorithms. Section 9.1.1 (*approximating the number of DNF satisfying assignments*) is based on [67], Section 9.1.2 (*finding perfect matching*) is based on [84], and Section 9.1.3 (*testing polynomial identities*) is based on [93, 115]. The *Randomized Rounding* technique was introduced in [89], and the *MaxSAT application* described in Section 9.1.4 is due to [52]. The *primality testing* algorithm described in Section 9.1.5 is folklore attributed to several people; I heard it attributed to M. Blum. Section 9.1.6 (*random walk algorithm for testing connectivity*) is based on [5], and Section 9.1.7 (*the randomized min-cut algorithm*) is based on [66].

Reductions (i.e., Complexity Theory). Section 9.2.1 (*reduction of approximate counting to deciding* and of *SAT to uniqueSAT*) is based on [96, 99] and [110], but the presentation in these sources is quite different. The reduction of Section 9.2.2 is based on [73], where it was used to show (independently of [96]) that $\mathcal{BPP} \in \mathcal{PH}$; the current presentation is due to Fortnow (priv. comm. 1997, see [10]). Section 9.2.3 (*self-corrector for the permanent*) is based on [76].

Protocols (i.e., Distributed Computing). Protocol 1 for *string equality* (in Section 9.3.1) is commonly attributed to M. Rabin and A. Yao, Protocol 2 is due to [85, Sec. 9], and Protocol 3 is due to E. Kushilevitz (priv. comm. 1998). Section 9.3.2 (*randomized routing*) is based on [107, 109], and Section 9.3.3 (*randomized Byzantine Agreement*) is based on [22, 87].

Deterministic alternatives. As mentioned in the main text, deterministic alternatives to randomized computations were found in two cases: (1) for the randomized primality tester presented in Section 9.1.5, and (2) for the randomized log-space connectivity tester presented in Section 9.1.6. Although the deterministic alternatives (of [1] and [90], respectively) are major breakthroughs, we believe that the original randomized computations are still of great interest.

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, Vol. 160 (2), pages 781–793, 2004.

[2] M. Ajtai. Generating Hard Instances of Lattice Problems. In *28th ACM Symposium on the Theory of Computing*, pages 99–108, 1996.

[3] M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

[4] M. Ajtai, J. Komlos, E. Szemerédi. An $O(n \log n)$ Sorting Network. In *15th ACM Symposium on the Theory of Computing*, pages 1–9, 1983.

[5] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[6] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[7] N. Alon, J. Bruck, J. Naor, M. Naor and R. Roth. Construction of Asymptotically Good, Low-Rate Error-Correcting Codes through Pseudo-Random Graphs. *IEEE Transactions on Information Theory*, Vol. 38, pages 509–516, 1992.

[8] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple Constructions of Almost k-wise Independent Random Variables. *Journal of Random Structures and Algorithms*, Vol. 3, No. 3, pages 289–304, 1992. Preliminary version in *31st FOCS*, 1990.

[9] N. Alon and J.H. Spencer. *The Probabilistic Method*. John Wiley & Sons, Inc., 1992. Second edition, 2000.

[10] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolin and L. Trevisan, Weak Random Sources, Hitting Sets, and BPP Simulations. To appear in *SIAM Journal on Computing*. Preliminary version in *38th FOCS*, 1997.

[11] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[12] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[13] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill, 1998.

[14] E. Bach and J. Shallit. *Algorithmic Number Theory* (Volume I: Efficient Algorithms). MIT Press, 1996.

[15] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling Algorithms: Lower Bounds and Applications. In *33rd ACM Symposium on the Theory of Computing*, pages 266–275, 2001.

[16] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in Interactive Proofs. *Computational Complexity*, Vol. 4 (4), pages 319–354, 1993. Extended abstract in *31st FOCS*, 1990.

[17] M. Bellare, O. Goldreich, and S. Goldwasser. Addendum to [16], available from `http://theory.lcs.mit.edu/~oded/papers.html`, May 1997.

[18] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-witnesses using an NP-oracle. *Information and Computation*, Vol. 163, pages 510–526, 2000.

[19] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

[20] M. Bellare, and J. Rompel. Randomness-efficient oblivious sampling. In *35th IEEE Symposium on Foundations of Computer Science*, 1994.

[21] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the Theory of Average Case Complexity. *Journal of Computer and System Science*, Vol. 44 (2), pages 193–219, 1992.

[22] M. Ben-Or. Another advantage of free choice: Completely Asynchronous Byzantine Agreement. In *2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[23] C.H. Bennett, G. Brassard and J.M. Robert. Privacy Amplification by Public Discussion. *SIAM Journal on Computing*, Vol. 17, pages 210–229, 1988. Preliminary version in *Crypto85*, Springer-Verlag Lecture Notes in Computer Science (Vol. 218), pages 468–476 (titled "How to Reduce your Enemy's Information").

[24] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, Vol. 36 (4), pages 889–974, 2006. Extended abstract in *36th STOC*, 2004.

[25] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. In *37th ACM Symposium on the Theory of Computing*, pages 266–275, 2005.

[26] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[27] R. Canetti, G. Even and O. Goldreich. Lower Bounds for Sampling Algorithms for Estimating the Average. *Information Processing Letters*, Vol. 53, pages 17–25, 1995.

[28] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.

[29] B. Chor and C. Dwork. Randomization in Byznatine Agreement. *Advances in Computing Research: A Research Annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 443–497, 1989.

[30] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[31] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17, No. 2, pages 230–261, 1988.

[32] B. Chor, J. Friedmann, O. Goldreich, J. Hastad, S. Rudich, and R. Smolansky. The Bit Extraction Problem or t-Resilient Functions. In *26th IEEE Symposium on Foundations of Computer Science*, pages 396–407, 1985.

[33] A. Cohen and A. Wigderson. Dispensers, Deterministic Amplification, and Weak Random Sources. In *30th IEEE Symposium on Foundations of Computer Science*, 1989, pages 14–19.

[34] T.M. Cover and G.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New-York, 1991.

[35] I. Dinur. The PCP Theorem by Gap Amplification. In *38th ACM Symposium on the Theory of Computing*, pages 241–250, 2006.

[36] .P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[37] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[38] G.D. Forney. *Concatenated Codes*. MIT Press, Cambridge, MA 1966.

[39] O. Gabber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *Journal of Computer and System Science*, Vol. 22, pages 407–420, 1981.

[40] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[41] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Algorithms and Combinatorics series (Vol. 17), Springer, 1999.

[42] O. Goldreich. *Foundation of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[43] O. Goldreich. Short Locally Testable Codes and Proofs (Survey). *ECCC*, TR05-014, 2005.

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

[45] O. Goldreich. On Testing Computability by Small Width OBDDs. *ECCC*, TR10-061, 2010.

[46] O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman. Security Preserving Amplification of Hardness. In *31st IEEE Symposium on Foundations of Computer Science*, pages 318–326, 1990.

[47] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[48] O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: the highly noisy case. *SIAM J. Discrete Math.*, Vol. 13 (4), pages 535–570, 2000.

[49] O. Goldreich, S. Vadhan and A. Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[50] O. Goldreich and A. Wigderson. Tiny Families of Functions with Random Properties: A Quality–Size Trade–off for Hashing. *Journal of Random structures and Algorithms*, Vol. 11, Nr. 4, December 1997, pages 315–343.

[51] S.W. Golomb. *Shift Register Sequences*. Holden-Day, 1967. (Aegean Park Press, revised edition, 1982.)

[52] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiablity problem. *SIAM Journal on Discrete Mathematics*, Vol. 7, No. 4, pages 656–666, 1994.

[53] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, Vol. 42, No. 6, 1995, pages 1115–1145.

[54] R. Graham, B. Rothschild, and J.H. Spencer. *Ramsey Theory*. John Wiley and Sons, 1990.

[55] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced Expanders and Randomness Extractors from Parvaresh-Vardy Codes. In *22nd IEEE Conference on Computational Complexity*, pages 96–108, 2007.

[56] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[57] J. Håstad. Getting optimal in-approximability results. *Journal of the ACM*, Vol. 48, pages 798–859, 2001. Extended abstract in *29th STOC*, 1997.

[58] A. Healy. Randomness-Efficient Sampling within NC1. *Computational Complexity*, to appear. Preliminary version in *10th RANDOM*, 2006.

[59] D. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems*. PWS, 1996.

[60] S. Hoory, N. Linial, and A. Wigderson. *Expander Graphs and their Applications. Bull. AMS*, Vol. 43 (4), pages 439–561, 2006.

[61] R. Impagliazzo, L.A. Levin and M. Luby. Pseudorandom Generation from One-Way Functions. In *21st ACM Symposium on the Theory of Computing*, pages 12–24, 1989.

[62] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *30th IEEE Symposium on Foundations of Computer Science*, 1989, pages 248–253.

[63] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Non-Negative Entries. *Journal of the ACM*, Vol. 51 (4), pages 671–697, 2004.

[64] M. Jerrum, L. Valiant, and V.V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, Vol. 43, pages 169–188, 1986.

[65] N. Kahale. Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

[66] D.R. Karger. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *4th SODA*, pages 21–30, 1993.

[67] R.M. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *24th IEEE Symposium on Foundations of Computer Science*, pages 56-64, 1983.

[68] R.M. Karp, N. Pippinger and M. Sipser. A Time-Randomness Tradeoff. *AMS Conference on Probabilistic Computational Complexity*, Durham, New Hampshire (1985).

[69] M. Karchmer and A. Wigderson. Monotone Circuits for Connectivity Require Superlogarithmic Depth. *SIAM J. Discrete Math.*, Vol. 3 (2), pages 255–265, 1990. Preliminary version in *20th STOC*, 1988.

[70] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *38th IEEE Symposium on Foundations of Computer Science*, 1997, pages 406–415.

[71] M.J. Kearns and U.V. Vazirani. *An introduction to Computational Learning Theory*. MIT Press, 1994.

[72] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996.

[73] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, Vol. 17, pages 215–217, 1983.

[74] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[75] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, August 1993.

[76] R.J. Lipton. New directions in testing. *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt (ed.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematics Society, Vol. 2, pages 191–202, 1991.

[77] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[78] C.-J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: optimal up to constant factors. In *35th ACM Symposium on the Theory of Computing*, pages 602–611, 2003.

[79] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[80] F. MacWilliams and N. Sloane. *The theory of error-correcting codes*. North-Holland, 1981.

[81] G.A. Margulis. Explicit Construction of Concentrators. *Prob. Per. Infor.*, Vol. 9 (4), pages 71–80, 1973 (in Russian). English translation in *Problems of Infor. Trans.*, pages 325–332, 1975.

[82] . Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005

[83] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[84] K. Mulmuley and U.V. Vazirani and V.V. Vazirani. Matching is as Easy as Matrix inversion. *Combinatorica*, Vol. 7, pages 105–113, 1987.

[85] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, Vol 22, 1993, pages 838–856. Preliminary version in *22nd STOC*, 1990.

[86] M.S. Paterson. Improved Sorting Networks with $O(\log N)$ Depth. *Algorithmica*, Vol. 5 (1), pages 75–92, 1990.

[87] M.O. Rabin. Randomized Byznatine Agreement. In *24th IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[88] J. Radhakrishnan and A. Ta-Shma: Bounds for Dispersers, Extractors, and Depth-Two Superconcentrators. *SIAM J. Discrete Math.*, Vol. 13 (1), pages 2–24, 2000.

[89] P. Raghavan and C.D. Thompson. Randomized Rounding. *Combinatorica*, Vol. 7, pages 365–374, 1987.

[90] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[91] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors. *Annals of Mathematics*, Vol. 155 (1), pages 157–187, 2001. Preliminary version in *41st FOCS*, pages 3–13, 2000.

[92] D. Ron. Algorithmic and Analysis Techniques in Property Testing. *Foundations and Trends in TCS*, Vol. 5 (2), pages 73–205, 2010.

[93] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, Vol. 27 (4), pages 701–717, October 1980.

[94] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. In *Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol 1: Algorithms and Complexity*, World scietific, 2004. (Editors: G. Paun, G. Rozenberg and A. Salomaa.) Preliminary version in *Bulletin of the EATCS 77*, pages 67–95, 2002.

[95] R. Shaltiel and C. Umans. Simple Extractors for All Min-Entropies and a New Pseudo-Random Generator. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[96] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[97] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, Vol. 6, pages 84–85, 1977. Addendum in *SIAM Journal on Computing*, Vol. 7, page 118, 1978.

[98] P.M. Spira. On time hardware complexity trade-offs for Boolean functions. In the *4th Hawaii International Symposium on System Sciences*, pages 525–527, 1971.

[99] L. Stockmeyer. The Complexity of Approximate Counting. In *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[100] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, Vol. 13 (1), pages 180–193, 1997.

[101] M. Sudan. Algorithmic introduction to coding theory. Lecture notes, Available from `http://theory.csail.mit.edu/~madhu/FT01/`, 2001.

[102] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, Vol. 20 (5), pages 865–877, 1991.

[103] L. Trevisan. When Hamming meets Euclid: The Approximability of Geometric TSP and MST. In *29th ACM Symposium on the Theory of Computing*, pages 21–29, 1997.

[104] L. Trevisan. Extractors and Pseudorandom Generators. *Journal of the ACM*, Vol. 48 (4), pages 860–879, 2001. Preliminary version in *31st STOC*, 1999.

[105] Y. Tzur. Notions of Weak Pseudorandomness and GF(2^n)-Polynomials. Master Thesis, Weizmann Institute of Science, 2009. Available from the theses section of *ECCC*.

[106] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[107] L.G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, Vol. 11 (2), pages 350–361, 1982.

[108] L.G. Valiant. Short Monotone Formulae for the Majority Function. *Journal of Algorithms*, Vol. 5 (3), pages 363–366, 1984.

[109] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *13th ACM Symposium on the Theory of Computing*, pages 263–277, 1981.

[110] L.G. Valiant and V.V. Vazirani. NP Is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, Vol. 47 (1), pages 85–93, 1986.

[111] U.V. Vazirani. *Randomness, Adversaries and Computation*. Ph.D. Thesis, EECS, UC Berkeley, 1986.

[112] E. Viola. The Sum of d Small-Bias Generators Fools Polynomials of Degree d. *Computational Complexity*, Vol. 18 (2), pages 209–217, 2009. Preliminary version in *23rd IEEE Conference on Computational Complexity*, 2008.

[113] A. Wigderson. The amazing power of pairwise independence. In *26th ACM Symposium on the Theory of Computing*, pages 645–647, 1994.

[114] S. Yekhanin. Towards 3-Query Locally Decodable Codes of Subexponential Length. In *39th ACM Symposium on the Theory of Computing*, pages 266–274, 2007.

[115] R. Zippel. Probabilistic algorithms for sparse polynomials. In the *Proceedings of EUROSAM '79: International Symposium on Symbolic and Algebraic Manipulation*, E. Ng (Ed.), Lecture Notes in Computer Science (Vol. 72), pages 216–226, Springer, 1979.

[116] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, Vol. 16, pages 367–391, 1996.

[117] D. Zuckerman. Randomness-Optimal Oblivious Sampling. *Journal of Random Structures and Algorithms*, Vol. 11, Nr. 4, December 1997, pages 345–367. Preliminary version in *28th STOC*, pages 286–295, 1996.

[118] D. Zuckerman. Linear-Degree Extractors and the Inapproximability of Max-Clique and Chromatic Number. In *38th ACM Symposium on the Theory of Computing*, 2006, pages 681–690.