Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–175

DOI: xxxxxx

now
the essence of knowledge

Pseudorandomness

Salil P. Vadhan[1]

[1] *Harvard UniversityCambridge, MA02138, USA, salil@eecs.harvard.edu*

Abstract

Pseudorandomness is the theory of efficiently generating objects that "look random" despite being constructed using little or no randomness. This survey of the subject places particular emphasis on the intimate connections that have been discovered between a variety of fundamental "pseudorandom objects" that at first seem very different in nature: expander graphs, randomness extractors, list-decodable error-correcting codes, samplers, and pseudorandom generators. The survey also illustrates the significance the theory of pseudorandomness has for the study of computational complexity, algorithms, cryptography, combinatorics, and communications. The structure of the presentation is meant to be suitable for teaching in a graduate-level course, with exercises accompanying each chapter.

To Kaya and Amari

Acknowledgments

My exploration of pseudorandomness began in my graduate and postdoctoral years at MIT and IAS, under the wonderful guidance of Oded Goldreich, Shafi Goldwasser, Madhu Sudan, and Avi Wigderson. It was initiated by an exciting reading group organized at MIT by Luca Trevisan, which immersed me in the subject and started my extensive collaboration with Luca. Through fortuitous circumstances, I also began to work with Omer Reingold, starting another lifelong collaboration. I am indebted to Oded, Shafi, Madhu, Avi, Luca, and Omer for all the insights and research experiences they have shared with me.

I have also learned a great deal from my other collaborators on pseudorandomness, including Boaz Barak, Eli Ben-Sasson, Michael Capalbo, Kai-Min Chung, Nenad Dedic, Dan Gutfreund, Venkat Guruswami, Iftach Haitner, Alex Healy, Jesse Kamp, Danny Lewin, Shachar Lovett, Chi-Jen Lu, Michael Mitzenmacher, Shien Jin Ong, Michael Rabin, Anup Rao, Ran Raz, Leo Reyzin, Eyal Rozenman, Madhur Tulsiani, Chris Umans, Emanuele Viola, Hoeteck Wee, and David Zuckerman. Needless to say, this list omits many other researchers in the field with whom I have had stimulating discussions.

The starting point for this survey was scribe notes taken by students in the 2004 version of my graduate course on Pseudorandomness. I thank those students for their contribution: Alexandr Andoni, Adi Akavia, Yan-Cheng Chang, Denis Chebikin, Hamilton Chong, Vitaly Feldman, Brian Greenberg, Chun-Yun Hsiao, Andrei Jorza, Adam Kirsch, Kevin Matulef, Mihai Pătraşcu, John Provine, Pavlo Pylyavskyy, Arthur Rudolph, Saurabh Sanghvi, Grant Schoenebeck, Jordanna Schutz, Sasha Schwartz, David Troiano, Vinod Vaikuntanathan, Kartik Venkatram, David Woodruff. I also thank the students from the other offerings of the course; Dan Gutfreund, who gave a couple of guest lectures in 2007; and all of my teaching fellows, Johnny Chen, Kai-Min Chung, Minh Nguyen, and Emanuele Viola. Special thanks are due to Greg Price and Dieter van Melkebeek for their extensive feedback on the lecture notes and draft of this survey, respectively. Helpful comments and corrections have also been given by Zachary Abel, Trevor Bass, Jeremy Booher, Zhou Fan, Alex Healy, Stephan Holzer, Andrei Jorza, Michael von Korff, Avner May, Eric Miles, Shira Mitchell, Jelani Nelson, Yakir Reshef, Shubhangi Saraf, Shrenik Shah, Madhu Sudan, Neal Wadhwa, and Hoeteck Wee.

Contents

1

Introduction

1.1 Overview of this Survey

Over the past few decades, *randomization* has become one of the most pervasive paradigms in computer science. Its widespread uses include:

Algorithm Design: For a number of important algorithmic problems, the most efficient algorithms known are randomized. For example:

- Primality Testing. This was shown to have a randomized polynomial-time algorithm in 1977. It wasn't until 2002 that a deterministic polynomial-time algorithm was discovered. (We will see this algorithm, but not its proof.)
- Approximate Counting. Many approximate counting problems (e.g. counting perfect matchings in a bipartite graph) have randomized polynomial-time algorithms, but the fastest known deterministic algorithms take exponential time.
- Undirected S-T Connectivity. This was shown to have a randomized logspace algorithm in 1979. It wasn't until 2005 that a deterministic logspace algorithm was discovered — using tools from the theory of pseudorandomness, as we will see.
- Perfect Matching. This was shown to have a randomized *polylogarithmic*-time parallel algorithm in the late 1970's. Deterministically, we only know polynomial-time algorithms.

Even in the cases where deterministic algorithms of comparable complexity were eventually found, the randomized algorithms remain considerably simpler and more efficient than the deterministic ones.

Cryptography: Randomization is central to cryptography. Indeed, cryptography is concerned with protecting secrets, and how can something be secret if it is deterministically fixed? For example, we assume that cryptographic keys are chosen at random (e.g. uniformly from the set of n-bit strings). In addition to the keys, it is known that often the cryptographic algorithms themselves

(e.g. for encryption) must be randomized to achieve satisfactory notions of security (e.g. that no partial information about the message is leaked).

Combinatorial Constructions: Randomness is often used to prove the *existence* of combinatorial objects with a desired property. Specifically, if one can show that a randomly chosen object has the property with nonzero probability, then it follows that such an object must, in fact, exist. A famous example due to Erdős is the existence of *Ramsey graphs*: A randomly chosen n-vertex graph has no clique or independent set of size $2 \log n$ with high probability. We will see several other applications of this "Probabilistic Method" in this survey, such as with two important objects mentioned below: expander graphs and error-correcting codes.

Though these *applications* of randomness are interesting and rich topics of study in their own right, they are not the focus of this survey. Rather, we ask the following:

Main Question: Can we reduce or even eliminate the use of randomness in these settings?

We have several motivations for doing this.

- Complexity Theory: We are interested in understanding and comparing the power of various kinds of computational resources. Since randomness is such a widely used resource, we want to know how it relates to other resources such as time, space, and parallelism. In particular, we ask: *Can every randomized algorithm be derandomized with only a small loss in efficiency?*
- Using Physical Random Sources: It is unclear whether the real world has physical sources of perfect randomness. We may use sources that seem to have some unpredictability, like the low order bits of a system clock or thermal noise, but these sources will generally have biases and, more problematically, correlations. Thus we ask: What can we do with a source of biased and correlated bits?
- Explicit Constructions: Probabilistic constructions of combinatorial objects often do not provide us with efficient algorithms for using those objects. Indeed, the randomly chosen object often has a description that is exponential in the relevant parameters. Thus, we look for *explicit* constructions — ones that are deterministic and efficient. In addition to their applications, improvements in explicit constructions serve as a measure of our progress in understanding the objects at hand. Indeed, Erdős posed the explicit construction of near-optimal Ramsey graphs as an open problem, and substantial progress on this problem was recently made using the theory of pseudorandomness (namely randomness extractors).
- Unexpected Applications: In addition, the theory of pseudorandomness has turned out to have many applications to problems that seem to have no connection to derandomization. These include data structures, distributed computation, circuit lower bounds and completeness results in complexity theory, reducing interaction in interactive protocols, saving memory in streaming algorithms, and more. We will see some of these applications in this survey (especially the exercises).

The paradigm we will use to study the Main Question is that of *pseudorandomness*: efficiently generating objects that "look random" using little or no randomness.

Specifically, we will study four "pseudorandom" objects:

Pseudorandom Generators: A pseudorandom generator is an algorithm that takes as input a short, perfectly random *seed* and then returns a (much longer) sequence of bits that "looks random." That the bits output cannot be perfectly random is clear — the output is determined by the seed and there are far fewer seeds than possible bit sequences. Nevertheless, it is possible for the output to "look random" in a very meaningful and general-purpose sense. Specifically, we will require that no *efficient* algorithm can distinguish the output from a truly random sequence. The study of pseudorandom generators meeting this strong requirement originated in cryptography, where they have numerous applications. In this survey, we will emphasize their role in derandomizing algorithms.

Note that asserting that a function is a pseudorandom generator is a statement about something that efficient algorithms can't do (in this case, distinguish two sequences). But proving that efficient algorithms cannot compute things is typically out of reach for current techniques theoretical computer science; indeed this is why the **P** vs. **NP** question is so hard. Thus, we will settle for conditional statements. An ideal theorem would be something like: "If $\mathbf{P} \neq \mathbf{NP}$, then pseudorandom generators exist." (The assumptions we make won't exactly be $\mathbf{P} \neq \mathbf{NP}$, but hypotheses of a similar flavor.)

Randomness Extractors: A randomness extractor takes as input a source of biased and correlated bits, and then produces a sequence of almost-uniform bits as output. Their original motivation was the simulation of randomized algorithms with sources of biased and correlated bits, but they have found numerous other applications in theoretical computer science. Ideally, extractors would be deterministic, but as we will see this proves to be impossible for general sources of biased and correlated bits. Nevertheless, we will get close — constructing extractors that are only "mildly" probabilistic, in that they use small (logarithmic) number of truly random bits as a seed for the extraction.

Expander Graphs: Expanders are graphs with two seemingly contradictory properties: they are sparse (e.g. having degree that is a constant, independent of the number of vertices), but also "well-connected" in some precise sense. For example, the graph cannot be bisected without cutting a large (say, constant) fraction of the edges.

Expander graphs have numerous applications in theoretical computer science. They were originally studied for their use in designing fault-tolerant networks (e.g. for telephone lines), ones that maintain good connectivity even when links or nodes fail. But they also have less obvious applications, such as an $O(\log n)$-time algorithm for sorting in parallel.

It is not obvious that expander graphs exist, but in fact it can be shown, via the Probabilistic Method, that a random graph of degree 3 is a "good" expander with high probability. However, many applications of expander graphs need *explicit constructions*, and these have taken longer to find. We will see some explicit constructions in this survey, but even the state-of-the-art does not always match the bounds given by the probabilistic method (in terms of the degree/expansion tradeoff).

Error-Correcting Codes: Error-correcting codes are tools for communicating over noisy channels. Specifically, they specify a way to encode messages into longer, redundant codewords so that even if the codeword gets somewhat corrupted along the way, it is still possible for the receiver to decode the original message. In his landmark paper that introduced the field of coding theory, Shannon also proved the existence of good error-correcting codes via the Probabilistic Method. That is, a random mapping of n-bit messages to $O(n)$-bit codewords is a "good" error-correcting code with high probability. Unfortunately, these probabilistic codes are not feasible to actually use — a random mapping requires an exponentially long description, and we know of no way to decode such a mapping efficiently. Again, explicit constructions are needed.

In this survey, we will focus on the problem of *list decoding*. Specifically, we will consider scenarios where the number of corruptions is so large that unique decoding is impossible; at best one can produce a short list that is guaranteed to contain the correct message.

A Unified Theory: Each of the above objects has been the center of a large and beautiful body of research, but until recently these corpora were largely distinct. An exciting development over the past decade has been the realization that all four of these objects are almost *the same* when interpreted appropriately. Their intimate connections will be a major focus of this survey, tying together the variety of constructions and applications that we will see.

The surprise and beauty of these connections has to do with the seemingly different nature of each of these objects. Pseudorandom generators, by asserting what efficient algorithms cannot do, are objects of complexity theory. Extractors, with their focus on extracting the entropy in a correlated and biased sequence, are information-theoretic objects. Expander graphs are of course combinatorial objects (as defined above), though they can also be interpreted algebraically, as we will see. Error-correcting codes involve a mix of combinatorics, information theory, and algebra. Because of the connections, we obtain new perspectives on each of the objects, and make substantial advances on our understanding of each by translating intuitions and techniques from the study of the others.

1.2 Background Required and Teaching Tips

The presentation assumes a good undergraduate background in the theory of computation, and general mathematical maturity. Specifically, it is assumed that the reader is familiar with basic algorithms and discrete mathematics, e.g. as covered in [CLRS], including some exposure to randomized algorithms; and with basic computational complexity including P, NP, and reductions, e.g. as covered in [Sip3]. Experience with elementary abstract algebra, particularly finite fields, is helpful; recommended texts are [Art, LN].

Most of the material in both volumes is covered in a one-semester graduate course that the author teaches at Harvard University, which consists of 24 lectures of 1.5 hours each. Most of the students in that course take at least one graduate-level course in theoretical computer science before this one.

The exercises are an important part of the survey, as they include proofs of some key facts, introduce some concepts that will be used in later chapters, and illustrate applications of the material to other topics. Problems that are particularly challenging or require more creativity than most are marked with a star.

1.3 Notational Conventions

We denote the set of numbers $\{1, \ldots, n\}$ by $[n]$. We write $\mathbb{N}$ for the set of nonnegative integers (so we consider 0 to be a natural number). We write $S \subset T$ to mean that S is a subset of T (not necessarily strict), and $S \subsetneq T$ for S being a strict subset of T. An inequality we use often is $\binom{n}{k} \leq (ne/k)^k$. All logarithms are base 2 unless otherwise specified. We often use the convention that lowercase letters are the logarithm (base 2) of the corresponding capital letter (e.g. $N = 2^n$).

Throughout, we consider random variables that can take values in arbitrary discrete sets (as well as real-valued random variables). We generally use capital letters, e.g. X, to denote random variables and lowercase letters, e.g. x, to denote specific values. We write $x \xleftarrow{R} X$ to indicate that x is sampled according to X. For a set S, we write $x \xleftarrow{R} S$ to mean that x is selected uniformly at random from S. We use the convention that multiple occurrences of a random variable in an expression refer to the same instantiation, e.g. $\Pr[X = X] = 1$. The *support* of a random variable X is denote $\mathrm{Supp}(X) \stackrel{\text{def}}{=} \{x : \Pr[X = x] > 0\}$. For an event E, we write $X|_E$ to denote the random variable X conditioned on the event E. For a set S, we write U_S to denote a random variable distributed uniformly over S. For $n \in \mathbb{N}$, U_n is a random variable distributed uniformly over $\{0,1\}^n$.

1.4 Chapter Notes and References

In this chapter, we only provide pointers to general surveys and textbooks on the topics covered, plus citations for specific results mentioned.

General introductions to the theory of pseudorandomness (other than this survey) include [Gol7, Mil2].

Recommended textbooks focused on randomized algorithms are [MU, MR]. The specific randomized and deterministic algorithms mentioned are due to [Mil1, Rab, SS2, AKS1, KLM, JSV, AKL$^+$, Rei, Lov1, KUW]; for more details see Section 2.6.

Recommended textbooks on cryptography are [Gol3, Gol4, KL]. The idea that encryption should be randomized is due to Goldwasser and Micali [GM].

The Probabilistic Method for combinatorial constructions is the subject of the book [AS1]. Erdős used this method to prove the existence of Ramsey graphs in [Erd]. Major recent progress on explicit constructions of Ramsey graphs was recently obtained by Barak, Rao, Shaltiel, and Wigderson [BRSW] via the theory of randomness extractors.

The modern notion of a pseudorandom generator was formulated in the works of Blum and Micali [BM] and Yao [Yao], motivated by cryptographic applications. We will spend most of our time on a variant of the Blum–Micali–Yao notion, proposed by Nisan and Wigderson [NW], where the generator is allowed more running time than the algorithms it fools. A detailed treatment of the Blum–Micali–Yao notion can be found in [Gol3].

Surveys on randomness extractors are [NT, Sha1]. The notion of extractor that we will focus on is the one due to Nisan and Zuckerman [NZ].

A detailed survey of expander graphs is [HLW]. The probabilistic construction of expander graphs is due to Pinsker [Pin]. The application of expanders to sorting in parallel is due to Ajtai, Komlós, and Szemerédi [AKS2].

A classic text on coding theory is [MS1]. For a modern, computer-science-oriented treatment, we recommend Sudan's lecture notes [Sud3]. Shannon started this field with the paper [Sha2]. The

notion of list decoding was proposed by Elias [Eli1] and Wozencraft [Woz], and was reinvigorated in the work of Sudan [Sud1]. Recent progress on list decoding is covered in [Gur2].

2

The Power of Randomness

Before we study the derandomization of randomized algorithms, we will need some algorithms to derandomize. Thus in this chapter, we present a number of examples of randomized algorithms, as well as develop the complexity-theoretic framework and basic tools for studying randomized algorithms.

2.1 Polynomial Identity Testing

In this section, we give a randomized algorithm to solve the following computational problem.

Computational Problem 2.1. Polynomial Identity Testing: given two multivariate polynomials, $f(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$, decide whether $f = g$.

This problem statement requires some clarification. Specifically, we need to say what we mean by:

- "polynomials": A *(multivariate) polynomial* is a finite expression of the form

$$f(x_1, \ldots, x_n) = \sum_{i_1, \ldots, i_n \in \mathbb{N}} c_{i_1, \ldots, i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}.$$

  We need to specify from what space the coefficients of the polynomials come; they could be the integers, reals, rationals, etc. In general, we will assume that the coefficients are chosen from a *field* (a set with addition and multiplication, where every nonzero element has a multiplicative inverse) or more generally an *(integral) domain* (where the product of two nonzero elements is always nonzero). Examples of fields include $\mathbb{Q}$ (the rationals), $\mathbb{R}$ (the reals), and $\mathbb{Z}_p$ (integers modulo p) where p is prime. An integral domain that is not a field is $\mathbb{Z}$ (the integers), but every integral domain is contained in its *field of fractions*, which is $\mathbb{Q}$ in the case of $\mathbb{Z}$. $\mathbb{Z}_n$ for composite n is not even an integral domain. We remark that there does exist a finite field $\mathbb{F}_q$ of size $q = p^k$ for every prime p and

positive integer k, and in fact this field is unique (up to isomorphism). Note that $\mathbb{F}_q$ is only equal to $\mathbb{Z}_q$ in case q is prime (i.e. $k = 1$). For more background on algebra, see the references in the chapter notes.

For a polynomial $f(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_n} c_{i_1,\ldots,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$, we define its *degree* (a.k.a. *total degree*) to be the maximum sum of the exponents $i_1 + \cdots + i_n$ over its monomials with nonzero coefficients $c_{i_1,\ldots,i_n}$. Its *degree in* x_j is the maximum of i_j over its monomials with nonzero coefficients.

- "$f = g$": What does it mean for two polynomials to be equal? There are two natural choices: the polynomials are the same as *functions* (they have the same output for every point in the domain), or the polynomials are the same as *formal polynomials* (the coefficients for each monomial are the same).

  These two definitions are equivalent over the integers (or more generally over infinite domains), but they are *not* equivalent over finite fields. For example, consider

  $$f(x) = \prod_{\alpha \in \mathbb{F}} (x - \alpha).$$

  for a finite field $\mathbb{F}$.[1] It is easy to see that $f(\alpha) = 0$ for all $\alpha \in \mathbb{F}$, but $f \neq 0$ as a formal polynomial. For us, *equality refers to equality as formal polynomials* unless otherwise specified.
- "given": What does it mean to be given a polynomial? There are several possibilities here:

  (1) As a list of coefficients: this trivializes the problem of Polynomial Identity Testing, as we can just compare. Since we measure complexity as a function of the input length, this algorithm runs in linear time.

  (2) As an "oracle": a black box that, given any point in the domain, gives the value of the polynomial. Here the only "explicit" inputs to the algorithm are the description of the field $\mathbb{F}$, the number n of variables, and a bound d on the degree of the polynomial, and we consider an algorithm to be efficient if it runs in time polynomial in the lengths of these inputs and $n \log |\mathbb{F}|$ (since it takes time at least $n \log |\mathbb{F}|$ to write down an input to the oracle).

  (3) As an *arithmetic formula*: a sequence of symbols like $(x_1+x_2)(x_3+x_7+6x_5)x_3(x_5-x_6) + x_2x_4(2x_3 + 3x_5)$ that describes the polynomial. Observe that while we can solve Polynomial Identity Testing by expanding the polynomials and grouping terms, the expanded polynomials may have length exponential in the length of the formula, and thus the algorithm is not efficient as a function of the input length.

  More general than formulas are circuits. An *arithmetic circuit* consists of a directed acyclic graph, consisting of *input nodes*, which have indegree 0 and are labeled by input variables or constants, and *computation nodes*, which have indegree 2 and are labelled by operations ($+$ or $\times$) specifying how to compute a value given the values at its children; one of the computation nodes is designated

[1] When expanded and terms are collected, this polynomial p can be shown to simply equal $x^{|\mathbb{F}|} - x$.

as the *output node*. Observe that every arithmetic circuit defines a polynomial in its input variables $x_1, \ldots, x_n$. Arithmetic formulas are equivalent to arithmetic circuits where the underlying graph is a tree.

The randomized algorithm we describe will work for all of the formulations above, but is only interesting for the second and third ones (oracles and arithmetic circuits/formulas). It will be convenient to work with the following equivalent version of the problem.

Computational Problem 2.2. Polynomial Identity Testing (reformulation): Given a polynomial $f(x_1, \ldots, x_n)$, is $f = 0$?

That is, we consider the special case of the original problem where $g = 0$. Any solution for the general version of course implies one for the special case; conversely, we can solve the general version by applying the special case to the polynomial $f' = f - g$.

Algorithm 2.3 (Polynomial Identity Testing**).**
Input: A multivariate polynomial $f(x_1, \ldots, x_n)$ of degree at most d over a field/domain $\mathbb{F}$.

(1) Let $S \subset \mathbb{F}$ be any set of size $2d$.
(2) Choose $\alpha_1, \ldots, \alpha_n \overset{\text{R}}{\leftarrow} S$.
(3) Evaluate $f(\alpha_1, \ldots, \alpha_n)$. If the result is 0, accept. Otherwise, reject.

It is clear that if $f = 0$, the algorithm will always accept. The correctness in case $f \neq 0$ is based on the following simple but very useful lemma.

Lemma 2.4 (Schwartz–Zippel Lemma). If f is a nonzero polynomial of degree d over a field (or integral domain) $\mathbb{F}$ and $S \subset \mathbb{F}$, then

$$\Pr_{\alpha_1, \ldots, \alpha_n \overset{\text{R}}{\leftarrow} S}[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}.$$

In the univariate case ($n = 1$), this amounts to the familiar fact that a polynomial with coefficients in a field and degree d has at most d roots. The proof for multivariate polynomials proceeds by induction on n, and we leave it as an exercise (Problem 2.1).

By the Schwartz-Zippel lemma, the algorithm will err with probability at most $1/2$ when $f \neq 0$. This error probability can be reduced by repeating the algorithm many times (or by increasing $|S|$). Note that the error probability is only over the coin tosses of the algorithm, not over the input polynomial f. This is what we mean when we say *randomized algorithm*; it should work on a worst-case input with high probability over the coin tosses of the algorithm. In contrast, algorithms whose correctness (or efficiency) only holds for inputs that are randomly chosen (according to a particular distribution) are called *heuristics*, and their study is called *average-case analysis*.

Note that we need a few things to ensure that our algorithm will work.

- First, we need a bound on the degree of the polynomial. We can get this in different ways depending on how the polynomial is represented. For example, for arithmetic formulas, the degree is bounded by the length of the formula. For arithmetic circuits, the degree is at most exponential in the size (or even depth) of the circuit.
- We also must be able to evaluate p when the variables take arbitrary values in some set S of size $2d$. For starters, this requires that the domain $\mathbb{F}$ is of size at least $2d$. We should also have an explicit description of the domain $\mathbb{F}$ enabling us to write down and manipulate field elements, and randomly sample from a subset of size at least $2d$ (e.g. the description of $\mathbb{F} = \mathbb{Z}_p$ is simply the prime p). Then, if we are given f as an oracle, we have the ability to evaluate f by definition. If we are given f as an arithmetic formula or circuit, then we can do a bottom-up, gate-by-gate evaluation. However, over infinite domains (like $\mathbb{Z}$), there is subtlety — the bit-length of the numbers can grow exponentially large. Problem 2.4 gives a method for coping with this.

Since these two conditions are satisfied, we have a polynomial-time randomized algorithm for Polynomial Identity Testing for polynomials given as arithmetic formulas over $\mathbb{Z}$ (or even circuits, by Problem 2.4). There are no known subexponential-time *deterministic* algorithms for this problem, even for formulas in $\Sigma\Pi\Sigma$ form (i.e. a sum of terms, each of which is the product of linear functions in the input variables). A deterministic polynomial-time algorithm for $\Sigma\Pi\Sigma$ formulas where the outermost sum has only a constant number of terms was obtained quite recently (2005).

2.1.1 Application to Perfect Matching

Now we will see an application of Polynomial Identity Testing to an important graph-theoretic problem.

Definition 2.5. Let $G = (V, E)$, then a *matching* on G is a set $E' \subset E$ such that no two edges in E' have a common endpoint. A *perfect matching* is a matching such that every vertex is incident to an edge in the matching.

Computational Problem 2.6. Perfect Matching: given a graph G, decide whether there is a perfect matching in G.

Unlike Polynomial Identity Testing, Perfect Matching is known to have deterministic polynomial-time algorithms — e.g. using alternating paths, or by reduction to Max Flow in the bipartite case. However, both of these algorithms seem to be inherently sequential in nature. With randomization, we can obtain an efficient parallel algorithm.

Algorithm 2.7 (Perfect Matching in bipartite graphs).
Input: a bipartite graph G with n vertices on each side.

We construct an $n \times n$ matrix A where

$$A_{i,j}(x) = \begin{cases} x_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases},$$

where $x_{i,j}$ is a formal variable.

Consider the multivariate polynomial

$$\det(A(x)) = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_i A_{i,\sigma(i)},$$

where S_n denotes the set of permutations on $[n]$. Note that the σ'th term is nonzero if and only if the permutation σ defines a perfect matching. That is, $(i, \sigma(i)) \in E$ for all $1 \leq i \leq n$. So $\det(A(x)) = 0$ iff G has no perfect matching. Moreover its degree is bounded by n, and given values $\alpha_{i,j}$ for the $x_{i,j}$'s we can evaluate $\det(A(\alpha))$ efficiently in parallel (in polylogarithmic time using an appropriate algorithm for the determinant.

So to test for a perfect matching efficiently in parallel, just run the Polynomial Identity Testing algorithm with, say, $S = \{1, \ldots, 2n\} \subset \mathbb{Z}$, to test whether $\det(A(x)) = 0$.

Some remarks:

- The above also provides the most efficient *sequential* algorithm for Perfect Matching, using the fact that Determinant has the same time complexity as Matrix Multiplication, which is known to be at most $O(n^{2.38})$.
- More sophisticated versions of the algorithm apply to nonbipartite graphs, and enable *finding* perfect matchings in the same parallel or sequential time complexity (where the result for sequential time is quite recent).
- Polynomial Identity Testing has been also used to obtain a randomized algorithm for Primality Testing, which was derandomized fairly recently (2002) to obtain the celebrated deterministic polynomial-time algorithm for Primality Testing. See Problem 2.5.

2.2 The Computational Model and Complexity Classes

2.2.1 Models of Randomized Computation

To develop a rigorous theory of randomized algorithms, we need to use a precise model of computation. There are several possible ways to augmenting a standard deterministic computational model (e.g. Turing machine or RAM model), such as:

(1) The algorithm has access to a "coin-tossing black box" that provides it with (unbiased and independent) random bits on request, with each request taking one time step. This is the model we will use.

(2) The algorithm has access to a black box that, given a number n in binary, returns a number chosen uniformly at random from $\{1, \ldots, n\}$. This model is often more convenient for describing algorithms. Problem 2.2 shows that it is equivalent to Model 1, in the sense that any problem that can be solved in polynomial time on one model can also be solved in polynomial time on the other.

(3) The algorithm is provided with an infinite tape (i.e. sequence of memory locations) that is initially filled with random bits. For polynomial-time algorithms, this is equivalent to Model 1. However, for space-bounded algorithms, this model seems stronger, as it provides the algorithm with free storage of its random bits (i.e. not counted towards its working memory). Model 1 is considered to be the "right" model for space-bounded algorithms. It is equivalent to allowing the algorithm *one-way* access to an infinite tape of random bits.

We interchangeable use "random bits" and "random coins" to refer to an algorithm's random choices.

2.2.2 Complexity Classes

We will now define complexity classes that capture the power of efficient randomized algorithms. As is common in complexity theory, these classes are defined in terms of decision problems, where the set of inputs where the answer should be "yes" is specified by a *language* $L \subset \{0,1\}^*$. However, the definitions generalize in natural ways to other types of computational problems, such as computing functions or solving search problems.

Recall that we say a deterministic algorithm A runs in time $t : \mathbb{N} \to \mathbb{N}$ if A takes at most $t(|x|)$ steps on every input x, and it runs in *polynomial time* if it runs time $t(n) = O(n^c)$ for a constant c. Polynomial time is a theoretical approximation to feasible computation, with the advantage that it is robust to reasonable changes in the model of computation and representation of the inputs.

Definition 2.8. **P** is the class of languages L for which there exists a deterministic polynomial-time algorithm A such that

- $x \in L \Rightarrow A(x)$ accepts.
- $x \notin L \Rightarrow A(x)$ rejects.

Here (and in the definitions below) the probabilities are taken over the coin tosses of the algorithm A.

For a randomized algorithm A, we say that A runs in time $t : \mathbb{N} \to \mathbb{N}$ if A takes at most $t(|x|)$ steps on every input x *and every sequence of random bits*.

Definition 2.9. **RP** is the class of languages L for which there exists a probabilistic polynomial-time algorithm A such that

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq 1/2$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

That is, **RP** algorithms may have *false negatives*; the algorithm may sometimes say "no" even if the answer is "yes", albeit with bounded probability. But the definition does not allow for false positives. Thus **RP** captures efficient randomized computation with *one-sided error*. **RP** stands

for "random polynomial time". Note that the error probability of an **RP** algorithm can be reduced to $2^{-p(n)}$ for any polynomial p by running the algorithm $p(n)$ times independently and accepting the input iff at least one of the trials accepts. By the same reasoning, the $1/2$ in the definition is arbitrary, and any constant $\alpha \in (0,1)$ or even $\alpha = 1/\text{poly}(n)$ would yield the same class of languages.

A central question in this survey is whether randomization enables us to solve more problems (e.g. decide more languages) in polynomial time:

Open Problem 2.10. Does $\mathbf{P} = \mathbf{RP}$?

Similarly, we can consider algorithms that may have false positives but no false negatives.

Definition 2.11. co-RP is the class of languages L whose complement $\bar{L}$ is in **RP**. Equivalently, $L \in$ **co-RP** if there exists a probabilistic polynomial-time algorithm A such that

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] = 1$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \leq 1/2$.

So, in **co-RP** we may err on NO instances, whereas in **RP** we may err on YES instances.

Using the POLYNOMIAL IDENTITY TESTING algorithm we saw earlier, we can deduce that POLYNOMIAL IDENTITY TESTING for arithmetic *formulas* is in **co-RP**. In Problem 2.4, this is generalized to arithmetic *circuits,* and thus we have:

Theorem 2.12. ARITHMETIC CIRCUIT IDENTITY TESTING over $\mathbb{Z}$, defined as the language

$$\text{ACIT}_{\mathbb{Z}} = \{C : C(x_1, \ldots, x_n) \text{ an arithmetic circuit over } \mathbb{Z} \text{ s.t. } C = 0\},$$

is in **co-RP**.

It is common to also allow two-sided error in randomized algorithms:

Definition 2.13. BPP is the class of languages L for which there exists a probabilistic polynomial-time algorithm A such that

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq 2/3$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \leq 1/3$.

Just as with **RP**, the error probability of **BPP** algorithms can be reduced from $1/3$ (or even $1/2 - 1/\text{poly}(n)$) to exponentially small by repetitions, this time taking a majority vote of the outcomes. Proving this uses some facts from probability theory, which we will review in the next section.

The cumbersome notation **BPP** stands for "bounded-error probabilistic polynomial-time," due to the unfortunate fact that **PP** ("probabilistic polynomial-time") refers to the definition where the

inputs in L are accepted with probability greater than $1/2$ and inputs not in L are accepted with probability at most $1/2$. Despite its name, **PP** is not a reasonable model for randomized algorithms, as it takes exponentially many repetitions to reduce the error probability. **BPP** is considered the standard complexity class associated with probabilistic polynomial-time algorithms, and thus the main question of this survey is:

Open Problem 2.14. Does $\mathbf{BPP} = \mathbf{P}$?

So far, we have considered randomized algorithms that can output an incorrect answer if they are unlucky in their coin tosses; these are called "Monte Carlo" algorithms. It is sometimes preferable to have "Las Vegas" algorithms, which always output the correct answer, but may run for a longer time if they are unlucky in their coin tosses. For this, we say that A has *expected running time* $t : \mathbb{N} \to \mathbb{N}$ if for every input x, the expectation of the number of steps taken by $A(x)$ is at most $t(|x|)$, where the expectation is taken over the coin tosses of A.

Definition 2.15. ZPP is the class of languages L for which there exists a probabilistic algorithm A that always decides L correctly and runs in expected polynomial time.

ZPP stands for "zero-error probabilistic polynomial time". The following relation between **ZPP** and **RP** is left as an exercise.

Fact 2.16 (Problem 2.3). $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{co\text{-}RP}$.

We do not know any other relations between the classes associated with probabilistic polynomial time.

Open Problem 2.17. Are any of the inclusions $\mathbf{P} \subset \mathbf{ZPP} \subset \mathbf{RP} \subset \mathbf{BPP}$ proper?

One can similarly define randomized complexity classes associated with complexity measures other than time such as space or parallel computation. For example:

Definition 2.18. RNC is the class of languages L for which there exists a probabilistic parallel algorithm A that runs in polylogarithmic time on polynomially many processors and such that

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq 1/2$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

A formal model of a parallel algorithm is beyond the scope of this survey, but can be found in standard texts on algorithms or parallel computation. We have seen:

Theorem 2.19. Perfect Matching in bipartite graphs, i.e. the language PM $= \{G :$ G a bipartite graph with a perfect matching$\}$, is in **RNC**.

Complexity classes associated with space-bounded computation will be discussed in Section 2.4.1.

2.2.3 Tail Inequalities and Error Reduction

In the previous section, we claimed that we can reduce the error of a **BPP** algorithm by taking independent repetitions and ruling by majority vote. The intuition that this should work is based on the Law of Large Numbers: if we repeat the algorithm many times, the fraction of correct answers should approach its expectation, which is greater than 1/2 (and thus majority rule will be correct). For complexity purposes, we need quantitative forms of this fact, which bound how many repetitions are needed to achieve a desired probability of correctness.

First, we recall a basic inequality which says that it is unlikely for (a single instantiation of) a random variable to exceed its expectation by a large factor.

Lemma 2.20 (Markov's Inequality). If X is a nonnegative real-valued random variable, then for any $\alpha > 0$,

$$\Pr[X \geq \alpha] \leq \frac{\mathrm{E}[X]}{\alpha}$$

Markov's Inequality alone does not give very tight concentration around the expectation; to get even a 50% probability, we need to look at deviations by a factor of 2. To get tight concentration, we can take many independent copies of a random variable. There are a variety of different tail inequalities that apply for this setting; they are collectively referred to as *Chernoff Bounds*.

Theorem 2.21 (A Chernoff Bound). Let $X_1, \ldots, X_t$ be independent random variables taking values in the interval $[0, 1]$, let $X = (\sum_i X_i)/t$, and $\mu = \mathrm{E}[X]$. Then

$$\Pr[|X - \mu| \geq \varepsilon] \leq 2\exp(-t\varepsilon^2/4).$$

Thus, the probability that the average deviates significantly from the expectation vanishes exponentially with the number of repetitions t. We leave the proof of this Chernoff Bound as an exercise (Problem 2.7).

Now let's apply the Chernoff Bound to analyze error-reduction for **BPP** algorithms.

Proposition 2.22. The following are equivalent:

(1) $L \in \mathbf{BPP}$.
(2) For every polynomial p, L has a probabilistic polynomial-time algorithm with two-sided error at most $2^{-p(n)}$.
(3) There exists a polynomial q such that L has a probabilistic polynomial-time algorithm with two-sided error at most $1/2 - 1/q(n)$.

Proof. Clearly, (2)$\Rightarrow$ (1) $\Rightarrow$ (3). Thus, we prove (3) $\Rightarrow$ (2).

Given an algorithm A with error probability at most $1/2 - 1/q(n)$, consider an algorithm A' that on an input x of length n, runs A for $t(n)$ independent repetitions and rules according to the majority, where $t(n)$ is a polynomial to be determined later.

We now compute the error probability of A' on an input x of length n. Let X_i be an indicator random variable that is 1 iff the i'th execution of $A(x)$ outputs the correct answer, and let $X = (\sum_i X_i)/t$ be the average of these indicators, where $t = t(n)$. Note that $A'(x)$ is correct when $X > 1/2$. By the error probability of A and linearity of expectations, we have $\mathrm{E}[X] \geq 1/2 + 1/q$, where $q = q(n)$. Thus, applying the Chernoff Bound with $\varepsilon = 1/q$, we have:

$$\Pr[X \leq 1/2] \leq 2 \cdot e^{-t/2q^2} < 2^{-p(n)},$$

for $t(n) = 2p(n)q(n)^2$ and sufficiently large n. □

2.3 Sampling and Approximation Problems

2.3.1 Sampling

The power of randomization is well-known to statisticians. If we want to estimate the mean of some quantity over a large population, we can do so very efficiently by taking the average over a small random sample.

Formally, here is the computational problem we are interested in solving.

Computational Problem 2.23. Sampling (a.k.a. [$+\varepsilon$]-Approx Oracle Average): Given oracle access to a function $f : \{0,1\}^m \to [0,1]$, estimate $\mu(f) \stackrel{\text{def}}{=} \mathrm{E}[f(U_m)]$ to within an additive error of ε. That is, output an answer in the interval $[\mu - \varepsilon, \mu + \varepsilon]$.

And here is the algorithm:

Algorithm 2.24 ([$+\varepsilon$]-Approx Oracle Average**).**
Input: Input: oracle access to a function $f : \{0,1\}^m \to [0,1]$

(1) Choose $x_1, \ldots, x_t \stackrel{\text{R}}{\leftarrow} \{0,1\}^m$, for an appropriate choice of $t = O(\log(1/\delta)/\varepsilon^2)$.
(2) Query the oracle to obtain $f(x_1), \ldots, f(x_t)$.
(3) Output $(\sum_i f(x_i))/t$.

The correctness of this algorithm follows from the Chernoff Bound (Theorem 2.21). Note that for constant ε and δ, the sample size t is independent of the size of the population (2^m), and we have running time $\mathrm{poly}(m)$ even for $\varepsilon = 1/\mathrm{poly}(m)$ and $\delta = 2^{-\mathrm{poly}(m)}$.

For this problem, we can *prove* that no deterministic algorithm can be nearly as efficient.

Proposition 2.25. Any deterministic algorithm solving [$+(1/4)$]-Approx Oracle Average must make at least $2^m/2$ queries to its oracle.

Proof. Suppose we have a deterministic algorithm A that makes fewer than $2^m/2$ queries. Let Q be the set of queries made by A when all of its queries are answered by 0. (We need to specify how the queries are answered to define Q, because A make its queries adaptively, with future queries depending on how earlier queries were answered.)

Now define two functions:

$$f_0(x) = 0 \quad \forall x$$
$$f_1(x) = \begin{cases} 0 & x \in Q \\ 1 & x \notin Q \end{cases}$$

Then A gives the same answer on both f_0 and f_1 (since all the oracle queries return 0 in both cases), but $\mu(f_0) = 0$ and $\mu(f_1) > 1/2$, so the answer must have error greater than $1/4$ for at least one of the functions. □

Thus, randomization provides an exponential savings for approximating the average of a function on a large domain. However, this does not show that **BPP** $\neq$ **P**. There are two reasons for this:

(1) [$+\varepsilon$]-Approx Oracle Average is not a decision problem, and indeed it is not clear how to define languages that capture the complexity of approximation problems. However, below we will see how a slightly more general notion of decision problem does allow us to capture approximation problems such as this one.
(2) More fundamentally, it does not involve the standard model of input as used in the definitions of **P** and **BPP**. Rather than the input being a string that is explicitly given to the algorithm (where we measure complexity in terms of the length of the string), the input is an exponential-sized oracle to which the algorithm is given random access. Even though this is not the classical notion of input, it is an interesting one that has received a lot of attention in recent years, because it allows for algorithms whose running time is sublimer (or even polylogarithmic) in the actual size of the input (e.g. 2^m in the example here). As in the example here, typically such algorithms require randomization and provide approximate answers.

2.3.2 Promise Problems

Now we will try to find a variant of the [$+\varepsilon$]-Approx Oracle Average problem that is closer to the **P** vs. **BPP** question. First, to obtain the standard notion of input, we consider functions that are presented in a concise form, as Boolean circuits $C : \{0,1\}^m \to \{0,1\}$ (analogous to the algebraic circuits defined in Section 2.1, but now the inputs take on Boolean values and the computation gates are $\wedge$, $\vee$, and $\neg$).

Next, we need a more general notion of decision problem than languages:

Definition 2.26. A *promise problem* Π consists of a pair (Π_Y, Π_N) of disjoint sets of strings, where Π_Y is the set of YES instances and Π_N is the set of NO instances. The corresponding computational problem is: given a string that is "promised" to be in Π_Y or Π_N, decide which is the case.

All of the complexity classes we have seen have natural promise-problem analogues, which we denote by **prP**, **prRP**, **prBPP**, etc. For example:

Definition 2.27. prBPP is the class of promise problems Π for which there exists a probabilistic polynomial-time algorithm A such that

- $x \in \Pi_Y \Rightarrow \Pr[A(x) \text{ accepts}] \geq 2/3$.
- $x \in \Pi_N \Rightarrow \Pr[A(x) \text{ accepts}] \leq 1/3$.

Since every language L corresponds to the promise problem $(L, \overline{L})$, any fact that holds for every promise problem in some promise-class also holds for every language in the corresponding language class. In particular, if every **prBPP** algorithm can be derandomized, so can every **BPP** algorithm:

Proposition 2.28. $\mathbf{prBPP} = \mathbf{prP} \Rightarrow \mathbf{BPP} = \mathbf{P}$.

Except where otherwise noted, all of the results in this survey about language classes (like **BPP**) easily generalize to the corresponding promise classes (like **prBPP**), but we state the results in terms of the language classes for notational convenience.

Now we consider the following promise problem.

Computational Problem 2.29. $[+\varepsilon]$-Approx Circuit Average is the promise problem $\mathrm{CA}^{\varepsilon}$, defined as:

$$\begin{aligned} \mathrm{CA}^{\varepsilon}_Y &= \{(C,p) : \mu(C) > p + \varepsilon\} \\ \mathrm{CA}^{\varepsilon}_N &= \{(C,p) : \mu(C) \leq p\} \end{aligned}$$

Here ε can be a constant or a function of the input length $n = |(C,p)|$.

It turns out that this problem completely captures the power of probabilistic polynomial-time algorithms.

Theorem 2.30. For every function ε such that $1/\mathrm{poly}(n) \leq \varepsilon(n) \leq 1 - 1/2^{n^{o(1)}}$, $[+\varepsilon]$-Approx Circuit Average is **prBPP**-complete. That is, it is in **prBPP** and every promise problem in **prBPP** reduces to it.

Proof Sketch. Membership in **prBPP** follows from Algorithm 2.24 and the fact that boolean circuits can be evaluated in polynomial time.

For **prBPP**-hardness, consider any promise problem $\Pi \in \mathbf{prBPP}$. We have a probabilistic $t(n)$-time algorithm A that decides Π with 2-sided error at most 2^{-n} on inputs of length n, where $t(n) = \mathrm{poly}(n)$. As an upper bound, A uses at most $t(n)$ random bits. *Thus we can view A as a deterministic algorithm on two inputs — its regular input $x \in \{0,1\}^n$ and its coin tosses $r \in \{0,1\}^{t(n)}$.* (This view of a randomized algorithm is useful throughout the study of pseudorandomness.) We'll write $A(x;r)$ for A's output on input x and coin tosses r. For every n, there is a circuit $C(x;r)$ of size $\mathrm{poly}(t(n)) = \mathrm{poly}(n)$ that simulates the computation of A for any input $x \in \{0,1\}^n$ $r \in \{0,1\}^{t(n)}$, and moreover C can be constructed in time $\mathrm{poly}(t(n)) = \mathrm{poly}(n)$. (See any text on complexity theory for a proof that circuits can efficiently simulate Turing machine computations.) Let $C_x(r)$ be the circuit C with x hardwired in. Then the map $x \mapsto (C_x, 1/2^n)$ is a polynomial-time reduction from Π to $[+\varepsilon]$-Approx Circuit Average. Indeed, if $x \in \Pi_N$, then A accepts with probability at most $1/2^n$, so $\mu(C_x) \leq 1/2^n$. And if $x \in \Pi_Y$, then $\mu(C_x) \geq 1 - 1/2^n > 1/2^n + \varepsilon(n')$, where $n' = |(C_x, 1/2^n)| = \mathrm{poly}(n)$ and we take n sufficiently large. $\square$

Consequently, derandomizing this one algorithm is equivalent to derandomizing all of **prBPP**:

Corollary 2.31. $[+\varepsilon]$-Approx Circuit Average is in **prP** if and only if **prBPP** = **prP**.

Note that our proof of Proposition 2.25, giving an exponential lower bound for $[+\varepsilon]$-Approx Oracle Average does not extend to $[+\varepsilon]$-Approx Circuit Average. Indeed, it's not even clear how to define the notion of "query" for an algorithm that is given a circuit C explicitly; it can do arbitrary computations that involve the internal structure of the circuit. Moreover, even if we restrict attention to algorithms that only use the input circuit C as if it were an oracle (other than computing the input length $|(C,p)|$ to know how long it can run), there is no guarantee that the function f_1 constructed in the proof of Proposition 2.25 has a small circuit.

2.3.3 Approximate Counting to within Relative Error

Note that $[+\varepsilon]$-Approx Circuit Average can be viewed as the problem of approximately counting the number of satisfying assignments of a circuit $C : \{0,1\}^m \to \{0,1\}$ to within additive error $\varepsilon \cdot 2^m$, and a solution to this problem may give useless answers for circuits that don't have very many satisfying assignments (e.g. circuits with fewer than $2^{m/2}$ satisfying assignments). Thus people typically study approximate counting to within *relative error*. For example, given a circuit C, output a number that is within a $(1+\varepsilon)$ factor of its number of satisfying assignments, $\#C$. Or the following essentially equivalent decision problem:

Computational Problem 2.32. $[\times(1+\varepsilon)]$-Approx #CSAT is the following promise problem:

$$\begin{aligned} \mathrm{CSAT}^{\varepsilon}_{Y} &= \{(C,N) : \#C > (1+\varepsilon)\cdot N\} \\ \mathrm{CSAT}^{\varepsilon}_{N} &= \{(C,N) : \#C \leq N\} \end{aligned}$$

Here ε can be a constant or a function of the input length $n = |(C,N)|$.

Unfortunately, this problem is **NP**-hard for general circuits (consider the special case that $N = 0$), so we do not expect a **prBPP** algorithm. However, there is a very pretty randomized algorithm if we restrict to DNF formulas.

Computational Problem 2.33. $[\times(1+\varepsilon)]$-Approx #DNF is the restriction of $[\times(1+\varepsilon)]$-Approx #CSAT to C that are formulas in *disjunctive normal form* (DNF) (i.e. an OR of clauses, where each clause is an AND of variables or their negations).

Theorem 2.34. For every function $\varepsilon(n) \geq 1/\mathrm{poly}(n)$, $[\times(1+\varepsilon)]$-Approx #DNF is in **prBPP**.

Proof. It suffices to give a probabilistic polynomial-time algorithm that estimates the number of satisfying assignments to within a $1 \pm \varepsilon$ factor. Let $\varphi(x_1, \ldots, x_m)$ be the input DNF formula.

A first approach would be to apply random sampling as we have used above: Pick t random assignments uniformly from $\{0,1\}^m$ and evaluate φ on each. If k of the assignments satisfy φ,

output $(k/t) \cdot 2^m$. However, if $\#\varphi$ is small (e.g. $2^{m/2}$), random sampling will be unlikely to hit any satisfying assignments, and our estimate will be 0.

The idea to get around this difficulty is to embed the set of satisfying assignments, A, in a smaller set B so that sampling can be useful. Specifically, we will define sets $A' \subset B$ satisfying the following properties:

(1) $|A'| = |A|$.
(2) $|A'| \geq |B|/\text{poly}(n)$, where $n = |(\varphi, N)|$.
(3) We can decide membership in A' in polynomial time.
(4) $|B|$ computable in polynomial time.
(5) We can sample uniformly at random from B in polynomial time.

Letting ℓ be the number of clauses, we define A' and B as follows:

$$\begin{aligned} B &= \Big\{(i,\alpha) \in [\ell] \times \{0,1\}^m : \alpha \text{ satisfies the } i^{\text{th}} \text{ clause}\Big\} \\ A' &= \Big\{(i,\alpha) \in B : \alpha \text{ does not satisfy any clauses before the } i^{\text{th}} \text{ clause}\Big\} \end{aligned}$$

Now we verify the desired properties:

(1) $|A| = |A'|$ because for each satisfying assignment α, A' contains only one pair (i, α), namely the one where the first clause satisfied by α is the i^{th} one.
(2) The size of A' and B can differ by at most a factor of ℓ, since each satisfying assignment α occurs at most ℓ times in B.
(3) It is easy to decide membership in A' in polynomial time.
(4) $|B| = \sum_{i=1}^{\ell} 2^{m-m_i}$, where m_i is the number of literals in clause i (after removing contradictory clauses that contain both a variable and its negation).
(5) We can randomly sample from B as follows. First pick a clause with probability proportional to the number of satisfying assignments it has (2^{m-m_i}). Then, fixing those variables in the clause (e.g. if x_j is in the clause, set $x_j = 1$, and if $\neg x_j$ is in the clause, set $x_j = 0$), assign the rest of the variables uniformly at random.

Putting this together, we deduce the following algorithm:

Algorithm 2.35 ($[\times(1+\varepsilon)]$-Approx #DNF**).**
Input: a DNF formula $\varphi(x_1, \ldots, x_m)$ with ℓ clauses

(1) Generate a random sample of t points in $B = \{(i,\alpha) \in [\ell] \times \{0,1\}^m : \alpha \text{ satisfies the } i^{\text{th}} \text{ clause}\}$, for an appropriate choice of $t = O(1/(\varepsilon/\ell)^2)$.
(2) Let $\hat{\mu}$ be the fraction of sample points that land in $A' = \{(i,\alpha) \in B : \alpha \text{ does not satisfy any clauses before the } i^{\text{th}} \text{ clause}\}$.
(3) Output $\hat{\mu} \cdot |B|$.

By the Chernoff bound, for an appropriate choice of $t = O(1/(\varepsilon/\ell)^2)$, we have $\hat{\mu} = |A'|/|B| \pm \varepsilon/\ell$ with high probability (where we write $\gamma = \alpha \pm \beta$ to mean that $\gamma \in [\alpha - \beta, \alpha + \beta]$). Thus, with high probability the output of the algorithm satisfies:

$$\begin{aligned} \hat{\mu} \cdot |B| &= |A'| \pm \varepsilon |B|/\ell \\ &= |A| \pm \varepsilon |A|. \end{aligned}$$

□

There is no deterministic polynomial-time algorithm known for this problem:

Open Problem 2.36. Give a deterministic polynomial-time algorithm for approximately counting the number of satisfying assignments to a DNF formula.

However, when we study pseudorandom generators in Chapter 7, we will see a quasipolynomial-time derandomization of the above algorithm (i.e. one in time $2^{\mathrm{polylog}(n)}$).

2.3.4 MaxCut

We give an example of another algorithm problem for which random sampling is a useful tool.

Definition 2.37. For a graph $G = (V, E)$ and $S, T \subset V$, define $\mathrm{cut}(S, T) = \{\{u, v\} \in E : u \in S, v \in T\}$, and $\mathrm{cut}(S) = \mathrm{cut}(S, V \setminus S)$.

Computational Problem 2.38. MaxCut (search version): Given G, find a largest cut in G, i.e. a set S maximizing $|\mathrm{cut}(S)|$.

Solving this problem optimally is **NP**-hard (in contrast to MinCut, which is known to be in **P**). However, there is a simple randomized algorithm that finds a cut of expected size at least $|E|/2$ (which is of course at least $1/2$ the optimal, and hence this is referred to as a "1/2-approximation algorithm"):

Algorithm 2.39 (MaxCut approximation).
Input: a graph $G = (V, E)$

Output a random subset $S \subset V$. That is, place each vertex v in S independently with probability $1/2$.

To analyze this algorithm, consider any edge $e = (u, v)$. Then the probability that e crosses the cut is $1/2$. By linearity of expectations, we have:

$$\mathrm{E}[|\mathrm{cut}(S)|] = \sum_{e \in E} \Pr[e \text{ is cut}] = |E|/2.$$

This also serves as a proof, via the probabilistic method, that every graph (without self-loops) has a cut of size at least $|E|/2$.

In Chapter 3, we will see how to derandomize this algorithm. We note that there is a much more sophisticated randomized algorithm that finds a cut whose expected size is within a factor of .878 of the largest cut in the graph (and this algorithm can also be derandomized).

2.4 Random Walks and S-T Connectivity

2.4.1 Graph Connectivity

One of the most basic problems in computer science is that of deciding connectivity in graphs:

Computational Problem 2.40. S-T Connectivity: Given a directed graph G and two vertices s and t, is there a path from s to t in G?

This problem can of course be solved in linear time using breadth-first or depth-first search. However, these algorithms also require linear space. It turns out that S-T Connectivity can in fact be solved using much less workspace. (When measuring the space complexity of algorithms, we do not count the space for the (read-only) input and (write-only) output.)

Theorem 2.41. There is an algorithm deciding S-T Connectivity using space $O(\log^2 n)$ (and time $O(n)^{\log n}$).

Proof. The following recursive algorithm $\text{IsPath}(G, u, v, k)$ decides whether there is a path of length at most k from u to v.

Algorithm 2.42 (Recursive S-T Connectivity).
$\text{IsPath}(G, u, v, k)$:

(1) If $k = 0$, accept if $u = v$.
(2) If $k = 1$, accept if $u = v$ or (u, v) is an edge in G.
(3) Otherwise, loop through all vertices w in G and accept if both $\text{IsPath}(G, u, w, \lceil k/2 \rceil)$ and $\text{IsPath}(G, w, v, \lfloor k/2 \rfloor)$ accept for some w.

We can solve S-T Connectivity by running $\text{IsPath}(G, s, t, n)$, where n is the number of vertices in the graph. The algorithm has $\log n$ levels of recursion and uses $\log n$ space per level of recursion (to store the vertex w), for a total space bound of $\log^2 n$. Similarly, the algorithm uses linear time per level of recursion, for a total time bound of $O(n)^{\log n}$. □

It is not known how to improve the space bound in Theorem 2.41 or to get the running time down to polynomial while maintaining space $n^{o(1)}$. For *undirected graphs*, however, we can do much better using a randomized algorithm. Specifically, we can place the problem in the following class:

Definition 2.43. A language L is in **RL** if there exists a randomized algorithm A that always halts, uses space at most $O(\log n)$ on inputs of length n, and satisfies:

- $x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq 1/2$.
- $x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0$.

Recall that our model of a randomized space-bounded machine is one that has access to a coin-tossing box (rather than an infinite tape of random bits), and thus must explicitly store in its workspace any random bits it needs to remember. The requirement that A always halts ensures that its running time is at most $2^{O(\log n)} = \text{poly}(n)$, because otherwise there would be a loop in its configuration space. Similarly to **RL**, we can define **L** (deterministic logspace), **co-RL** (one-sided error with errors only on NO instances), and **BPL** (two-sided error).

Now we can state the theorem for undirected graphs.

Computational Problem 2.44. UNDIRECTED S-T CONNECTIVITY: Given an undirected graph G and two vertices s and t, is there a path from s to t in G?

Theorem 2.45. UNDIRECTED S-T CONNECTIVITY is in **RL**.

Proof Sketch. The algorithm simply does a polynomial-length random walk starting at s.

Algorithm 2.46 (UNDIRECTED S-T CONNECTIVITY via Random Walks).
Input: (G, s, t), where $G = (V, E)$ has n vertices.

(1) Let $v = s$.
(2) Repeat $\text{poly}(n)$ times:
  (a) If $v = t$, halt and accept.
  (b) Else let $v \stackrel{R}{\leftarrow} \{w : (v, w) \in E\}$.
(3) Reject (if we haven't visited t yet).

Notice that this algorithm only requires space $O(\log n)$, to maintain the current vertex v as well as a counter for the number of steps taken. Clearly, it never accepts when there isn't a path from s to t. In the next section, we will prove that in any connected undirected graph, a random walk of length $\text{poly}(n)$ from one vertex will hit any other vertex with high probability. Applying this to the connected component containing s, it follows that the algorithm accepts with high probability when s and t are connected. □

This algorithm, dating from the 1970's, was derandomized only in 2005. We will cover the derandomized algorithm in Section 4.4. However, the general question of derandomizing space-bounded algorithms remains open.

Open Problem 2.47. Does $\mathbf{RL} = \mathbf{L}$? Does $\mathbf{BPL} = \mathbf{L}$?

2.4.2 Random Walks on Graphs

For generality that will be useful later, many of the definitions in this section will be given for *directed multigraphs* (which we will refer to as *digraphs* for short). By multigraph, we mean that we allow G to have parallel edges and self-loops. Henceforth, we will refer to graphs without parallel edges and self-loops as *simple graphs*. We call a digraph *d-regular* if every vertex has indegree d and outdegree d. To analyze the random-walk algorithm of the previous section, it suffices to prove a bound on the *hitting time* of random walks.

Definition 2.48. For a digraph $G = (V, E)$, we define its *hitting time* as

$$\mathrm{hit}(G) = \max_{i,j \in V} \min\{t : \Pr[\text{a random walk of length } t \text{ started at } i \text{ visits } j] \geq 1/2\}.$$

We note that $\mathrm{hit}(G)$ is often defined as the maximum over vertices i and j of the *expected* time for a random walk from i to visit j. The two definitions are the same up to a factor of 2, and the above is more convenient for our purposes.

We will prove:

Theorem 2.49. For every connected undirected graph G with n vertices and maximum degree d, we have $\mathrm{hit}(G) = O(d^2 n^3 \log n)$.

We observe that it suffices to prove this theorem for *d-regular* graphs, because any undirected graph be made regular by adding self-loops (and this can only increase the hitting time). This is the part of our proof that fails for directed graphs (adding self-loops cannot correct an imbalance between indegree and outdegree at a vertex), and indeed Problem 2.10 shows that general directed graphs can have exponential hitting time.

There are combinatorial methods for proving the above theorem, but we will prove it using a linear-algebraic approach, as the same methods will be very useful in our study of expander graphs. For an n-vertex digraph G, we define its *random-walk transition matrix*, or *random-walk matrix* for short, to be the $n \times n$ matrix M where $M_{i,j}$ is the probability of going from vertex i to vertex j in one step. That is, $M_{i,j}$ is the number of edges from i to j divided by the outdegree of i. In case G is d-regular, M is simply the adjacency matrix of G divided by d. Notice that for every probability distribution $\pi \in \mathbb{R}^n$ on the vertices of G (written as a row vector), the vector πM is the probability distribution obtained by selecting a vertex i according to π and then taking one step of the random walk to end at a vertex j. This is because $(\pi M)_j = \sum_i \pi_i M_{i,j}$.

In our application to bounding $\mathrm{hit}(G)$ for a *regular* digraph G, we start at a probability distribution $\pi = (0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{R}^n$ concentrated at vertex i, and are interested in the distribution πM^t we get after taking t steps on the graph. Specifically, we'd like to show that it places nonnegligible mass on vertex j for $t = \mathrm{poly}(n)$. We will do this by showing that it in fact converges to the uniform distribution $u = (1/n, 1/n, \ldots, 1/n) \in \mathbb{R}^n$ within a polynomial number of steps. Note that

$uM = u$ by the regularity of G, so convergence to u is possible (and will be guaranteed given some additional conditions on G).

We will measure the rate of convergence in the ℓ_2 norm. For vectors $x, y \in \mathbb{R}^n$, we will use the standard inner product $\langle x, y\rangle = \sum_i x_i y_i$, and ℓ_2 norm $\|x\| = \sqrt{\langle x, x\rangle}$. We write $x \perp y$ to mean that x and y are orthogonal, i.e. $\langle x, y\rangle = 0$. We want to determine how large k needs to be so that $\|\pi M^k - u\|$ is "small". This is referred to as the *mixing time* of the random walk. Mixing time can be defined with respect to various distance measures and the ℓ_2 norm is not the most natural one, but it has the advantage that we will be able to show that the distance decreases noticeably in every step. This is captured by the following quantity.

Definition 2.50. For a regular digraph G with random-walk matrix M, we define

$$\lambda(G) \stackrel{\text{def}}{=} \max_{\pi} \frac{\|\pi M - u\|}{\|\pi - u\|} = \max_{x \perp u} \frac{\|xM\|}{\|x\|},$$

where the first maximization is over all *probability distributions* $\pi \in [0,1]^n$ and the second is over all vectors $x \in \mathbb{R}^n$ such that $x \perp u$. We write $\gamma(G) \stackrel{\text{def}}{=} 1 - \lambda(G)$.

To see that the first definition of $\lambda(G)$ is smaller than or equal to the second, note that for any probability distribution π, the vector $x = (\pi - u)$ is orthogonal to uniform (i.e. the sum of its entries is zero). For the converse, observe that given any vector $x \perp u$, the vector $\pi = u + \alpha x$ is a probability distribution for a sufficiently small α. It can be shown that $\lambda(G) \in [0,1]$. (This follows from Problems 2.10 and 2.9.)

The following lemma is immediate from the definition of $\lambda(G)$.

Lemma 2.51. Let G be a regular digraph with random-walk matrix M. For every initial probability distribution π on the vertices of G and every $t \in \mathbb{N}$, we have

$$\|\pi M^t - u\| \leq \lambda(G)^t \cdot \|\pi - u\| \leq \lambda(G)^t.$$

Proof. The first inequality follows from the definition of $\lambda(G)$. For the second, we have:

$$\begin{aligned}
\|\pi - u\|^2 &= \|\pi^2\| + \|u\|^2 - 2\langle \pi, u\rangle \\
&= \sum_i \pi_i^2 + 1/N - 2/N \sum_i \pi_i \\
&\leq \sum_i \pi_i + 1/N - 2/N \sum_i \pi_i \\
&= 1 - 1/N \leq 1.
\end{aligned}$$

□

Thus a smaller value of $\lambda(G)$ (equivalently, a larger value of $\gamma(G)$) means that the random walk mixes more quickly. Specifically, for $t \geq \ln(n/\varepsilon)/\gamma(G)$, it follows that every entry of πM^t has probability mass at least $1/n - (1 - \gamma(G))^t \geq (1 - \varepsilon)/n$, and thus we should think of the walk as being "mixed" within $O((\log n)/\gamma(G))$. Note that after $O(1/\gamma(G))$ steps, the ℓ_2 distance is already

small (say at most 1/10), but this is not a satisfactory notion of mixing — a distribution that assigns ε^2 mass to $1/\varepsilon^2$ vertices has ℓ_2 distance smaller than ε from the uniform distribution.

So the mixing time of the random walk on G is at most $O((\log n)/\gamma(G))$, and this holds with respect to any reasonable distance measure. Note that $O(1/\gamma(G))$ steps does not suffice, because a distribution with ℓ_2 distance ε from uniform could just assign equal probability mass to $1/\varepsilon^2$ vertices (and thus be very far from uniform in any intuitive sense).

Corollary 2.52. For every regular digraph G, $\text{hit}(G) = O(n \log n/\gamma(G))$.

Proof. Let i, j be any two vertices of G. As argued above, a walk of length $\ln(2n)/\gamma(G)$ from any start vertex has a probability of at least $1/2n$ of ending at j. Thus, if we do $O(n)$ such walks consecutively, we will hit j with probability at least $1/2$. □

Thus, our task reduces to bounding $\gamma(G)$:

Theorem 2.53. If G is a connected, nonbipartite, and d-regular undirected graph on n vertices, then $\gamma(G) = \Omega(1/(dn)^2)$.

Theorem 2.53 and Corollary 2.52 imply Theorem 2.49, as any undirected and connected graph of maximum degree d can be made $(d+1)$-regular and nonbipartite by adding self-loops to each vertex. (We remark that the bounds of Theorems 2.53 and 2.49 are not tight.) Theorem 2.53 is proven in Problem 2.9, using a connection with eigenvalues described in the next section.

2.4.3 Eigenvalues

Recall that a nonzero vector $v \in \mathbb{R}^n$ is an *eigenvector* of $n \times n$ matrix M if $vM = \lambda v$ for some $\lambda \in \mathbb{R}$, which is called the corresponding *eigenvalue*. A useful feature of *symmetric* matrices is that they can be described entirely in terms of their eigenvectors and eigenvalues.

Theorem 2.54 (Spectral Theorem for Symmetric Matrices). If M is a symmetric $n \times n$ real matrix with distinct eigenvalues $\mu_1, \ldots, \mu_k$, then the eigenspaces $W_i = \{v \in \mathbb{R}^n : vM = \mu_i M\}$ are orthogonal (i.e. $v \in W_i$, $w \in W_j \Rightarrow v \perp w$ if $i \neq j$) and span $\mathbb{R}^n$ (i.e. $\mathbb{R}^n = W_1 + \cdots + W_k$). We refer to the dimension of W_i as the *multiplicity* of eigenvalue μ_i. In particular, $\mathbb{R}^n$ has a basis consisting of orthogonal eigenvectors $v_1, \ldots, v_n$ having respective eigenvalues $\lambda_1, \ldots, \lambda_n$, where the number of times μ_i occurs among the λ_j's exactly equals the multiplicity of μ_i.

Notice that if G is a undirected regular graph, then its random-walk matrix M is symmetric. We know that $uM = u$, so the uniform distribution is an eigenvector of eigenvalue 1. Let $v_2, \ldots, v_n$ and $\lambda_2, \ldots, \lambda_n$ be the remaining eigenvectors and eigenvalues, respectively. Given any probability distribution π, we can write it as $\pi = u + c_2 v_2 + \cdots + c_n v_n$. Then the probability distribution after k steps on the random walk is

$$\pi M^t = u + \lambda_2^t c_2 v_2 + \cdots + \lambda_n^t c_n v_n.$$

In Problem 2.9, it is shown that all of the λ_i's have absolute value at most 1. Notice that if they all have magnitude strictly smaller than 1, then πM^t indeed converges to u. Thus it is not surprising that our measure of mixing rate, $\lambda(G)$, equals the absolute value of the second largest eigenvalue.

Lemma 2.55. Let G be an undirected graph with random-walk matrix M. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of M, sorted so that $1 = \lambda_1 \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$. Then $\lambda(G) = |\lambda_2|$.

Proof. Let $u = v_1, v_2, \ldots, v_n$ be the basis of orthogonal eigenvectors corresponding to the λ_i's. Given any vector $x \perp u$, we can write $x = c_2 v_2 + \cdots + c_n v_n$. Then:

$$\begin{aligned} \|xM\|^2 &= \|\lambda_2 c_2 v_2 + \cdots + \lambda_n c_n v_n\|^2 \\ &= \lambda_2^2 c_2^2 \|v_2\|^2 + \cdots + \lambda_n^2 c_n^2 \|v_n\|^2 \\ &\leq |\lambda_2|^2 \cdot (c_2^2 \|v_2\|^2 + \cdots + c_n^2 \|v_n\|^2) \\ &= |\lambda_2|^2 \cdot \|x\|^2 \end{aligned}$$

Equality is achieved with $x = v_2$. □

Thus, bounding $\lambda(G)$ amounts to bounding the eigenvalues of G. Due to this connection, $\gamma(G) = 1 - \lambda(G)$ is often referred to as the *spectral gap*, as it is the gap between the largest eigenvalue and the second largest eigenvalue in absolute value.

2.4.4 Markov Chain Monte Carlo

Random walks are a powerful tool in the design of randomized algorithms. In particular, they are the heart of the "Markov Chain Monte Carlo" method, which is widely used in statistical physics and for solving approximate counting problems. In these applications, the goal is to generate a random sample from an *exponentially* large space, such as an (almost) uniformly random perfect matching for a given bipartite graph G. (It turns out that this is equivalent to approximately counting the number of perfect matchings in G.) The approach is to do a random walk on an appropriate (regular) graph $\hat{G}$ defined on the space (e.g. by doing random local changes on the current perfect matching). Even though $\hat{G}$ is typically of size exponential in the input size $n = |G|$, in many cases it can be proven to have mixing time $\text{poly}(n) = \text{polylog}(|\hat{G}|)$, a property referred to as *rapid mixing*. These Markov Chain Monte Carlo methods provide some of the best examples of problems where randomization yields algorithms that are exponentially faster than all known deterministic algorithms.

2.5 Exercises

~~Problem 2.1 (Schwartz–Zippel lemma).~~ ~~Prove Lemma 2.4: If $f(x_1, \ldots, x_n)$ is a nonzero polynomial of degree d over a field (or integral domain) $\mathbb{F}$ and $S \subset \mathbb{F}$, then~~

$$\Pr_{\alpha_1, \ldots, \alpha_n \overset{R}{\leftarrow} S}[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}.$$

You may use the fact that every nonzero *univariate* polynomial of degree d over $\mathbb{F}$ has at most d roots.

Problem 2.2 (Robustness of the model). Suppose we modify our model of randomized computation to allow the algorithm to obtain a random element of $\{1, \ldots, m\}$ for any number m whose binary representation it has already computed (as opposed to just allowing it access to random bits). Show that this would not change the classes **BPP** and **RP**.

Problem 2.3 (Zero error vs. 1-sided error). Prove that $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{co\text{-}RP}$.

Problem 2.4 (Polynomial Identity Testing for integer circuits). In this problem, you will show how to do Polynomial Identity Testing for arithmetic *circuits* over the integers. The Prime Number Theorem says that the number of primes less than T is $(1 \pm o(1)) \cdot T/\ln T$, where the $o(1)$ tends to 0 as $T \to \infty$. You may use this fact in the problem below.

(1) Show that if N is a nonzero integer and $M \stackrel{\text{R}}{\leftarrow} \{1, \ldots, \log^2 N\}$, then

$$\Pr[N \not\equiv 0 \pmod{M}] = \Omega(1/\text{loglog}N).$$

(2) Use the above to prove Theorem 2.12: Arithmetic Circuit Identity Testing over $\mathbb{Z}$ is in **co-RP**.

Problem 2.5 (Polynomial Identity Testing via Modular Reduction). In this problem, you will analyze an alternative to the algorithm seen in class, which directly handles polynomials of degree larger than the field size. It is based on the same idea as Problem 2.4, using the fact that polynomials over a field have many of the same algebraic properties as the integers.

The following definitions and facts may be useful: A polynomial $f(x)$ over a field $\mathbb{F}$ is called *irreducible* if it has no nontrivial factors (i.e. factors other than constants from $\mathbb{F}$ or constant multiples of f). Analogously to prime factorization of integers, every polynomial over $\mathbb{F}$ can be factored into irreducible polynomials and this factorization is unique (up to reordering and constant multiples). It is known that the number of irreducible polynomials of degree at most d over a field $\mathbb{F}$ is at least $|\mathbb{F}|^{d+1}/2d$. (This is similar to the Prime Number Theorem for integers mentioned in Problem 2.4, but is easier to prove.) For polynomials $f(x)$ and $g(x)$, $f(x) \bmod g(x)$ is the remainder when f is divided by h. (More background on polynomials over finite fields can be found in the references listed in Section 2.6.)

In this problem, we consider a version of the Polynomial Identity Testing problem where a polynomial $f(x_1, \ldots, x_n)$ over finite field $\mathbb{F}$ is presented as a formula built up from elements of $\mathbb{F}$ and the variables $x_1, \ldots, x_n$ using addition, multiplication, and *exponentiation* with exponents given in *binary*. We also assume that we are given a representation of $\mathbb{F}$ enabling addition, multiplication, and division in $\mathbb{F}$ to be done quickly.

(1) Let $f(x)$ be a univariate polynomial of degree $\leq D$ over a field $\mathbb{F}$. Prove that there are constants c, c' such that if $f(x)$ is nonzero (as a formal polynomial) and $g(x)$ is a randomly selected polynomial of degree at most $d = c \log D$, then the probability that $f(x) \bmod h(x)$ is nonzero is at least $1/c' \log D$. Deduce a randomized, polynomial-time identity test for *univariate* polynomials presented in the above form.
(2) Obtain an identity test for multivariate polynomials by reduction to the univariate case.

Problem 2.6 (Primality Testing). (1) Show that for every positive integer n, the polynomial identity $(x+1)^n \equiv x^n + 1 \pmod{n}$ holds iff n is prime.
(2) Obtain a **co-RP** algorithm for the language Primality Testing$= \{n : n \text{ prime}\}$ using Part 1 together with Problem 2.5. (In your analysis, remember that the integers modulo n are a field only when n is prime.)

Problem 2.7 (Chernoff Bound). Let $X_1, \ldots, X_t$ be independent $[0,1]$-valued random variables, and $X = \sum_{i=1}^{t} X_i$. (Note that, in contrast to the statement of Theorem 2.21, here we are writing X for the sum of the X_i's rather than their average.)

(1) Show that for every $r \in [0, 1/2]$, $\mathrm{E}[e^{rX}] \leq e^{r\,\mathrm{E}[X] + r^2 t}$. (Hint: $1 + x \leq e^x \leq 1 + x + x^2$ for all $x \in [0, 1/2]$.)
(2) Deduce the Chernoff Bound of Theorem 2.21: $\Pr[X \geq \mathrm{E}[X] + \varepsilon t] \leq e^{-\varepsilon^2 t/4}$ and $\Pr[X \leq \mathrm{E}[X] - \varepsilon t] \leq e^{-\varepsilon^2 t/4}$.
(3) Where did you use the independence of the X_i's?

Problem 2.8 (Necessity of Randomness for Identity Testing*). In this problem, we consider the "oracle version" of the identity testing problem, where an arbitrary polynomial $f : \mathbb{F}^m \to \mathbb{F}$ of degree d is given as an oracle and the problem is to test whether $f = 0$. Show that any deterministic algorithm that solves this problem when $m = d = n$ must make at least 2^n queries to the oracle (in contrast to the randomized identity testing algorithm from class, which makes only one query provided that $|\mathbb{F}| \geq 2n$).

Is this a proof that $\mathbf{P} \neq \mathbf{RP}$? Explain.

Problem 2.9 (Spectral Graph Theory). Let M be the random-walk matrix for a d-regular *undirected* graph $G = (V, E)$ on n vertices. We allow G to have self-loops and multiple edges. Recall that the uniform distribution is an eigenvector of M of eigenvalue $\lambda_1 = 1$. Prove the following statements. (Hint: for intuition, it may help to think about what the statements mean for the behavior of the random walk on G.)

(1) All eigenvalues of M have absolute value at most 1.

(2) G is disconnected $\iff$ 1 is an eigenvalue of multiplicity at least 2.
(3) Suppose G is connected. Then G is bipartite $\iff$ -1 is an eigenvalue of M.
(4) G connected $\Rightarrow$ all eigenvalues of M other than λ_1 are at most $1 - 1/\text{poly}(n, d)$. To do this, it may help to first show that the second largest eigenvalue of M (not necessarily in absolute value) equals

$$\max_x \langle xM, x\rangle = 1 - \frac{1}{d} \cdot \min_x \sum_{(i,j) \in E} (x_i - x_j)^2,$$

where the maximum/minimum is taken over all vectors x of length 1 such that $\sum_i x_i = 0$, and $\langle x, y\rangle = \sum_i x_i y_i$ is the standard inner product. For intuition, consider restricting the above maximum/minimum to $x \in \{+\alpha, -\beta\}^n$ for $\alpha, \beta > 0$.
(5) G connected and nonbipartite $\Rightarrow$ all eigenvalues of M (other than 1) have absolute value at most $1 - 1/\text{poly}(n, d)$ and thus $\gamma(G) \geq 1/\text{poly}(n, d)$.
(6*) Establish the (tight) bound $1 - \Omega(1/d \cdot D \cdot n)$ in Part 4, where D is the diameter of the graph. Conclude that $\gamma(G) = \Omega(1/d^2n^2)$ if G is connected and nonbipartite.

Problem 2.10 (Hitting Time and Eigenvalues for Directed Graphs).
(1) Show that for every n, there exists a digraph G with n vertices, outdegree 2, and $\text{hit}(G) = 2^{\Omega(n)}$.
(2) Let G be a *regular* digraph with random-walk matrix M. Show that $\lambda(G) = \sqrt{\lambda(G')}$, where G' is the *undirected* graph whose random-walk matrix is MM^T.
(3) A digraph G is called *Eulerian* if it is connected and every vertex has the same indegree as outdegree.[2] (For example, if we take an connected undirected graph and replace each undirected edge $\{u, v\}$ with the two directed edges (u, v) and (v, u), we obtain an Eulerian digraph.) Show that if G is an n-vertex Eulerian digraph of maximum degree d, then $\text{hit}(G) = \text{poly}(n, d)$.

2.6 Chapter Notes and References

Recommended textbooks on randomized algorithms are Motwani–Raghavan [MR] and Mitzenmacher–Upfal [MU]. The algorithmic power of randomization became apparent in the 1970's with a number of striking examples, notably Berlekamp's algorithm for factoring polynomials over large finite fields [Ber3] and the Miller–Rabin [Mil1, Rab] and Solovay–Strassen [SS2] algorithms for Primality Testing. The randomized algorithm for Polynomial Identity Testing was independently discovered by DeMillo and Lipton [DL], Schwartz [Sch], and Zippel [Zip]. A deterministic polynomial-time Polynomial Identity Testing algorithm for formulas in $\Sigma\Pi\Sigma$ form with a constant number of terms was given by Kayal and Saxena [KS], improving a previous quasipolynomial-time algorithm of Dvir and Shpilka [DS]. Problem 2.4 is from [Sch]. Problem 2.8 is from [LV].

Recommended textbooks on abstract algebra and finite fields are [Art, LN].

[2] This terminology comes from the fact that these are precisely the digraphs that have *Eulerian circuits,* closed paths that visit all vertices use every directed edge exactly once.

The randomized algorithm for PERFECT MATCHING is due to Lovász, who also showed how to extend the algorithm to nonbipartite graphs. An efficient parallel randomized algorithm for *finding* a perfect matching was given by Karp, Upfal, and Wigderson [KUW] (see also [MVV]). A randomized algorithm for finding a perfect matching in the same sequential time complexity as Lovász's algorithm was given recently by Mucha and Sankowski [MS] (see also [Har]).

The efficient sequential and parallel algorithms for DETERMINANT mentioned in the text are due [Str, CW2] and [Csa, BvzGH, Ber1], respectively. For more on algebraic complexity and parallel algorithms, we refer to the textbooks [BCS] and [Lei], respectively. The POLYNOMIAL IDENTITY TESTING and PRIMALITY TESTING algorithms of Problems 2.5 and 2.6 are due to Agrawal and Biswas [AB]. Agrawal, Kayal, and Saxena [AKS1] derandomized the PRIMALITY TESTING algorithm to prove that PRIMALITY TESTING is in **P**.

The randomized complexity classes **RP**, **BPP**, **ZPP**, and **PP** were formally defined by Gill [Gil2], who conjectured that $\mathbf{BPP} \neq \mathbf{P}$ (in fact $\mathbf{ZPP} \neq \mathbf{P}$). Chernoff Bounds are named after H. Chernoff [Che2]; the version in Theorem 2.21 is due to Hoeffding [Hoe] and is sometimes referred to as Hoeffding's Inequality. The monograph by Dubhashi and Panconesi [DP] has a detailed coverage of Chernoff Bounds and other tail inequalities. Problem 2.3 is due to Rabin (cf. [Gil2]).

The computational perspective on sampling, as introduced in Section 2.3.1, is surveyed in [Gol1, Gol2]. SAMPLING is perhaps the simplest example of a computational problem where randomization enables algorithms with running time sublinear in the size of the input. Such *sublinear-time algorithms* are now known for a wide variety of interesting computational problems; see the surveys [Ron, Rub].

Promise problems were introduced by Even, Selman, and Yacobi [ESY]. For survey of their role in complexity theory, see Goldreich [Gol5].

The randomized algorithm for $[\times(1+\varepsilon)]$-APPROX #DNF is due to Karp and Luby [KLM], who initiated the study of randomized algorithms for approximate counting problems. A 1/2-approximation algorithm for MAXCUT was first given in [SG]; that algorithm can be viewed as a natural derandomization of Algorithm 2.39. (See Algorithm 3.17.) The .878-approximation algorithm was given by Goemans and Williamson [GW].

The $O(\log^2 n)$-space algorithm for S-T CONNECTIVITY is due to Savitch [Sav]. Using the fact that S-T CONNECTIVITY (for *directed* graphs) is complete for nondeterministic logspace (**NL**), this result is equivalent to the fact that $\mathbf{NL} \subset \mathbf{L}^2$, where $\mathbf{L}^c$ is the class of languages that can be decided deterministic space $O(\log^c n)$. The latter formulation (and its generalization $\mathbf{NSPACE}(s(n)) \subset \mathbf{DSPACE}(s(n)^2)$) is known as Savitch's Theorem. The randomized algorithm for UNDIRECTED S-T CONNECTIVITY was given by Aleliunas, Karp, Lipton, Lovász, and Rackoff [AKL$^+$], and was recently derandomized by Reingold [Rei] (see Section 4.4).

For more background on random walks, mixing time, and the Markov Chain Monte Carlo Method, we refer the reader to [MU, MR, Ran]. The use of this method for counting perfect matchings is due to [Bro, JS, JSV].

The bound on hitting time given in Theorem 2.49 is not tight; for example, it can be improved to $\Theta(n^2)$ for regular graphs that are simple (have no self-loops or parallel edges) [KLNS].

Even though we will focus primarily on undirected graphs (for example, in our study of expanders in Chapter 4), much of what we do generalizes to regular, or even Eulerian, digraphs. See Problem 2.10 and the references [Mih, Fil, RTV]. Problem 2.10, Part 2 is from [Fil].

The Spectral Theorem (Thm. 2.54) can be found in any standard textbook on linear algebra. Problem 2.9 is from [Lov3]; Alon and Sudakov [AS2] strengthen it to show $\gamma(G) = \Omega(1/dDn)$, which implies a tight bound of $\gamma(G) = \Omega(1/n^2)$ for simple graphs (where $D = O(n/d)$).

Spectral Graph Theory is a rich subject, with many applications beyond the scope of this text; see the survey by Spielman [Spi] and references therein.

One significant omission from this chapter is the usefulness of randomness for *verifying proofs*. Recall that **NP** is the class of languages having membership proofs that can be verified in **P**. Thus it is natural to consider proof verification that is probabilistic, leading to the class **MA**, as well as a larger class **AM**, where the proof itself can depend on the randomness chosen by the verifier [BM]. (These are both subclasses of the class **IP** of languages having *interactive proof systems* [GMR].) There are languages, such as Graph Nonisomorphism, that are in **AM** but are not known to be in **NP** [GMW]. "Derandomizing" these proof systems (e.g. proving **AM** = **NP**) would show that Graph Nonisomorphism is in **NP**, i.e. that there are short proofs that two graphs are nonisomorphic. Similarly to sampling and sublinear-time algorithms, randomized proof verification can also enable one to read only a small portion of an appropriately encoded **NP** proof, leading to the celebrated PCP Theorem and its applications to hardness of approximation [FGL$^+$, AS, ALM$^+$]. For more about interactive proofs and PCPs, see [Vad, Gol6, AB].

3

Basic Derandomization Techniques

In the previous chapter, we saw some striking examples of the power of randomness for the design of efficient algorithms:

- Polynomial Identity Testing in **co-RP**.
- $[\times(1+\varepsilon)]$-Approx #DNF in **prBPP**.
- Perfect Matching in **RNC**.
- Undirected S-T Connectivity in **RL**.
- Approximating MaxCut in probabilistic polynomial time.

This is of course only a small sample; there are entire texts on randomized algorithms. (See the notes and references for Chapter 2.)

In the rest of this survey, we will turn towards *derandomization* — trying to remove the randomness from these algorithms. We will achieve this for some of the specific algorithms we studied, and also consider the larger question of whether *all* efficient randomized algorithms can be derandomized, e.g. does $\mathbf{BPP} = \mathbf{P}$? $\mathbf{RL} = \mathbf{L}$? $\mathbf{RNC} = \mathbf{NC}$?

In this chapter, we will introduce a variety of "basic" derandomization techniques. These will each be deficient in that they are either infeasible (e.g. cannot be carried out in polynomial time) or specialized (e.g. apply only in very specific circumstances). But it will be useful to have these as tools before we proceed to study more sophisticated tools for derandomization (namely, the "pseudorandom objects" of Chapters 4–7).

3.1 Enumeration

We are interested in quantifying how much savings randomization provides. One way of doing this is to find the smallest possible upper bound on the deterministic time complexity of languages in **BPP**. For example, we would like to know which of the following complexity classes contain **BPP**:

Definition 3.1 (Deterministic Time Classes). [1]

$$\begin{aligned}
\mathbf{DTIME}(t(n)) &= \{L : L \text{ can be decided deterministically in time } O(t(n))\} \\
\mathbf{P} &= \cup_c \mathbf{DTIME}(n^c) && \text{(“polynomial time”)} \\
\tilde{\mathbf{P}} &= \cup_c \mathbf{DTIME}(2^{(\log n)^c}) && \text{(“quasipolynomial time”)} \\
\mathbf{SUBEXP} &= \cap_\varepsilon \mathbf{DTIME}(2^{n^\varepsilon}) && \text{(“subexponential time”)} \\
\mathbf{EXP} &= \cup_c \mathbf{DTIME}(2^{n^c}) && \text{(“exponential time”)}
\end{aligned}$$

The “Time Hierarchy Theorem” of complexity theory implies that all of these classes are distinct, i.e. $\mathbf{P} \subsetneq \tilde{\mathbf{P}} \subsetneq \mathbf{SUBEXP} \subsetneq \mathbf{EXP}$. More generally, it says that $\mathbf{DTIME}(o(t(n)/\log t(n))) \subsetneq \mathbf{DTIME}(t(n))$ for any efficiently computable time bound t. (What is difficult in complexity theory is separating classes that involve different computational resources, like deterministic time vs. nondeterministic time.)

Enumeration is a derandomization technique that enables us to deterministically simulate any randomized algorithm with an exponential slowdown.

Proposition 3.2. $\mathbf{BPP} \subset \mathbf{EXP}$.

Proof. If L is in **BPP**, then there is a probabilistic polynomial-time algorithm A for L running in time $t(n)$ for some polynomial t. As in the proof of Theorem 2.30, we write $A(x;r)$ for A's output on input $x \in \{0,1\}^n$ and coin tosses $r \in \{0,1\}^{m(n)}$, where we may assume $m(n) \leq t(n)$ without loss of generality. Then:

$$\Pr[A(x;r) \text{ accepts}] = \frac{1}{2^{m(n)}} \sum_{r \in \{0,1\}^{m(n)}} A(x;r)$$

We can compute the right-hand side of the above expression in deterministic time $2^{m(n)} \cdot t(n)$. □

We see that the enumeration method is *general* in that it applies to all **BPP** algorithms, but it is *infeasible* (taking exponential time). However, if the algorithm uses only a small number of random bits, it becomes feasible:

Proposition 3.3. If L has a probabilistic polynomial-time algorithm that runs in time $t(n)$ and uses $m(n)$ random bits, then $L \in \mathbf{DTIME}(t(n) \cdot 2^{m(n)})$. In particular, if $t(n) = \text{poly}(n)$ and $m(n) = O(\log n)$, then $L \in \mathbf{P}$.

Thus an approach to proving $\mathbf{BPP} = \mathbf{P}$ is to show that the number of random bits used by any **BPP** algorithm can be reduced to $O(\log n)$. We will explore this approach in Chapter 7. However, to date, Proposition 3.2 remains the best unconditional upper-bound we have on the deterministic time-complexity of **BPP**.

[1] Often $\mathbf{DTIME}(\cdot)$ is written as $\mathbf{TIME}(\cdot)$, but we include the **D** to emphasize the it refers to deterministic rather than randomized algorithms.

Open Problem 3.4. Is **BPP** "closer" to **P** or **EXP**? Is **BPP** $\subset \tilde{\mathbf{P}}$? Is **BPP** $\subset$ **SUBEXP**?

3.2 Nonconstructive/Nonuniform Derandomization

Next we look at a derandomization technique that can be implemented efficiently but requires some nonconstructive "advice" that depends on the input length.

Proposition 3.5. If $A(x; r)$ is a randomized algorithm for a language L that has error probability smaller than 2^{-n} on inputs x of length n, then for every n, there exists a fixed sequence of coin tosses r_n such that $A(x; r_n)$ is correct for all $x \in \{0,1\}^n$.

Proof. We use the Probabilistic Method. Consider R_n chosen uniformly at random from $\{0,1\}^{r(n)}$, where $r(n)$ is the number of coin tosses used by A on inputs of length n. Then

$$\begin{aligned} \Pr[\exists x \in \{0,1\}^n \text{ s.t. } A(x; R_n) \text{ incorrect on } x] &\leq \sum_x \Pr[A(x; R_n) \text{ incorrect on } x] \\ &< 2^n \cdot 2^{-n} = 1 \end{aligned}$$

Thus, there exists a fixed value $R_n = r_n$ that yields a correct answer for all $x \in \{0,1\}^n$. □

The advantage of this method over enumeration is that once we have the fixed string r_n, computing $A(x; r_n)$ can be done in polynomial time. However, the proof that r_n exists is nonconstructive; it is not clear how to find it in less than exponential time.

Note that we know that we can reduce the error probability of any **BPP** (or **RP**, **RL**, **RNC**, etc.) algorithm to smaller than 2^{-n} by repetitions, so this proposition is always applicable. However, we begin by looking at some interesting special cases.

Example 3.6 (Perfect Matching). We apply the proposition to Algorithm 2.7. Let $G = (V, E)$ be a bipartite graph with m vertices on each side, and let $A^G(x_{1,1}, \ldots, x_{m,m})$ be the matrix that has entries $A^G_{i,j} = x_{i,j}$ if $(i, j) \in E$, and $A^G_{i,j} = 0$ if $(i, j) \notin E$. Recall that the polynomial $\det(A^G(x))$ is nonzero if and only if G has a perfect matching. Let $S_m = \{0, 1, 2, \ldots, m2^{m^2}\}$. We argued that, by the Schwartz–Zippel Lemma, if we choose $\alpha \xleftarrow{R} S_m^{m^2}$ at random and evaluate $\det(A^G(\alpha))$, we can determine whether $\det(A^G(x))$ is zero with error probability at most $m/|S|$ which is smaller than 2^{-m^2}. Since a bipartite graph with m vertices per side is specified by a string of length $n = m^2$, by Proposition 3.5 we know that for every m, there exists an $\alpha_m \in S_m^{m^2}$ such that $\det(A^G(\alpha)) \neq 0$ if and only if G has a perfect matching, for every bipartite graph G with m vertices on each side.

Open Problem 3.7. Can we find such an $\alpha_m \in \{0, \ldots, m2^{m^2}\}^{m^2}$ explicitly, i.e., *deterministically* and *efficiently*? An **NC** algorithm (i.e. parallel time polylog(m) with poly(m) processors) would put Perfect Matching in deterministic **NC**, but even a subexponential-time algorithm would be interesting.

Example 3.8 (Universal Traversal Sequences). Let G be a connected d-regular undirected multigraph on n vertices. From Theorem 2.49, we know that a random walk of $\text{poly}(n, d)$ steps from any start vertex will visit any other vertex with high probability. By increasing the length of the walk by a polynomial factor, we can ensure that *every* vertex is visited with probability greater than $1 - 2^{-nd \log n}$. By the same reasoning as in the previous example, we conclude that for every pair (n, d), there exists a *universal traversal sequence* $w \in \{1, 2, \ldots, d\}^{\text{poly}(n,d)}$ such that for every n-vertex, d-regular, connected G and every vertex s in G, if we start from s and follow w then we will visit the entire graph.

Open Problem 3.9. Can we construct such a universal traversal sequence explicitly (e.g. in polynomial time or even logarithmic space)?

There has been substantial progress towards resolving this question in the positive; see Section 4.4.

We now cast the nonconstructive derandomizations provided by Proposition 3.5 in the language of "nonuniform" complexity classes.

Definition 3.10. Let $\mathbf{C}$ be a class of languages, and $\mathbf{a} : \mathbb{N} \to \mathbb{N}$ be a function. Then $\mathbf{C/a}$ is the class of languages defined as follows: $L \in \mathbf{C/a}$ if there exists $L' \in \mathbf{C}$, and $\alpha_1, \alpha_2, \ldots \in \{0,1\}^*$ with $|\alpha_n| \leq a(n)$, such that $x \in L \Leftrightarrow (x, \alpha_{|x|}) \in L'$. The α's are called the *advice* strings.

$\mathbf{P/poly}$ is the class $\bigcup_c \mathbf{P}/n^c$, i.e. polynomial time with polynomial advice.

A basic result in complexity theory is that $\mathbf{P/poly}$ is exactly the class of languages that can be decided by polynomial-sized Boolean circuits:

Fact 3.11. $L \in \mathbf{P/poly}$ iff there is a sequence of Boolean circuits $\{C_n\}_{n \in \mathbb{N}}$ and a polynomial p such that for all n

(1) $C_n : \{0,1\}^n \to \{0,1\}$ decides $L \cap \{0,1\}^n$
(2) $|C_n| \leq p(n)$.

We refer to $\mathbf{P/poly}$ as a "nonuniform" model of computation because it allows for different, unrelated "programs" for each input length (e.g. the circuits C_n, or the advice α_n), in contrast to classes like $\mathbf{P}$, $\mathbf{BPP}$, and $\mathbf{NP}$, that require a single "program" of constant size specifying how the computation should behave for inputs of arbitrary length. Although $\mathbf{P/poly}$ contains some undecidable problems,[2] people generally believe that $\mathbf{NP} \not\subset \mathbf{P/poly}$. Indeed, trying to prove lower bounds on circuit size is one of the main approaches to proving $\mathbf{P} \neq \mathbf{NP}$, since circuits seem much more concrete and combinatorial than Turing machines. However, this too has turned out to be quite difficult; the best circuit lower bound known for computing an explicit function is roughly $5n$.

Proposition 3.5 directly implies:

[2] Consider the unary version of the HALTING PROBLEM, which can be decided in constant time given advice $\alpha_n \in \{0,1\}$ that specifies whether the n'th Turing machine halts or not.

Corollary 3.12. $\mathbf{BPP} \subset \mathbf{P/poly}$.

A more general meta-theorem is that "nonuniformity is more powerful than randomness."

3.3 Nondeterminism

Although physically unrealistic, nondeterminism has proven to be a very useful resource in the study of computational complexity (e.g. leading to the class **NP**). Thus it is natural to study how it compares in power to randomness. Intuitively, with nondeterminism we should be able to guess a "good" sequence of coin tosses for a randomized algorithm and then do the computation deterministically. This intuition does apply directly for randomized algorithms with 1-sided error:

Proposition 3.13. $\mathbf{RP} \subset \mathbf{NP}$.

Proof. Let $L \in \mathbf{RP}$ and A be a randomized algorithm that decides it. A polynomial-time verifiable witness that $x \in L$ is any sequence of coin tosses r such that $A(x; r) = \texttt{accept}$. □

However, for 2-sided error (**BPP**), containment in **NP** is not clear. Even if we guess a 'good' random string (one that leads to a correct answer), it is not clear how we can verify it in polynomial time. Indeed, it is consistent with current knowledge that **BPP** equals **NEXP** (nondeterministic exponential time)! Nevertheless, there is a sense in which we can show that **BPP** is no more powerful than **NP**:

Theorem 3.14. If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{P} = \mathbf{BPP}$.

Proof. For any language $L \in \mathbf{BPP}$, we will show how to express membership in L using two quantifiers. That is, for some polynomial-time predicate P,

$$x \in L \iff \exists y \forall z \, P(x, y, z), \tag{3.1}$$

where we quantify over y and z of length $\text{poly}(|x|)$.

Assuming $\mathbf{P} = \mathbf{NP}$, we can replace $\forall z \, P(x, y, z)$ by a polynomial-time predicate $Q(x, y)$, because the language $\{(x, y) : \forall z \, P(x, y, z)\}$ is in $\mathbf{co\text{-}NP} = \mathbf{P}$. Then $L = \{x : \exists y \, Q(x, y)\} \in \mathbf{NP} = \mathbf{P}$.

To obtain the two-quantifier expression (3.1), consider a randomized algorithm A for L, and assume w.l.o.g. that its error probability is smaller than 2^{-n} and that it uses $m = \text{poly}(n)$ coin tosses. Let $A_x \subset \{0,1\}^m$ be the set of coin tosses r for which $A(x; r) = 0$. We will show that if x is in L, there exist m "shifts" (or "translations") of A_x that cover all points in $\{0,1\}^m$. (Notice that this is a $\exists\forall$ statement.) Intuitively, this should be possible because A_x contains all but an exponentially small fraction of $\{0,1\}^m$. On the other hand if $x \notin L$, then no m shifts of A_x can cover all of $\{0,1\}^m$. Intuitively, this is because A_x is an exponentially small fraction of $\{0,1\}^m$.

Formally, by a "shift of A_x" we mean a set of the form $A_x \oplus s = \{r \oplus s : r \in A_x\}$ for some

$s \in \{0,1\}^m$; note that $|A_x \oplus s| = |A_x|$. We will show

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \exists s_1, s_2, \ldots, s_m \in \{0,1\}^m \forall r \in \{0,1\}^m \quad r \in \bigcup_{i=1}^{m} (A_x \oplus s_i) \\ &\Leftrightarrow \quad \exists s_1, s_2, \ldots, s_m \in \{0,1\}^m \, \forall r \in \{0,1\}^m \quad \bigvee_{i=1}^{m} (A(x; r \oplus s_i) = 1) \, ; \end{aligned}$$

$$\begin{aligned} x \notin L \quad &\Rightarrow \quad \forall s_1, s_2, \ldots, s_m \in \{0,1\}^m \, \exists r \in \{0,1\}^m \quad r \notin \bigcup_{i=1}^{m} (A_x \oplus s_i) \\ &\Leftrightarrow \quad \forall s_1, s_2, \ldots, s_m \in \{0,1\}^m \, \exists r \in \{0,1\}^m \quad \neg \bigvee_{i=1}^{m} (A(x; r \oplus s_i) = 1) \, . \end{aligned}$$

We prove both parts by the Probabilistic Method, starting with the second (which is simpler).

$x \notin L$ Let $s_1, \ldots, s_m$ be arbitrary, and choose $R \stackrel{\text{R}}{\leftarrow} \{0,1\}^m$ at random. Now A_x and hence each $A_x \oplus s_i$ contains less than a 2^{-n} fraction of $\{0,1\}^m$. So, by a union bound,

$$\begin{aligned} \Pr[R \in \bigcup_i (A_x \oplus s_i)] &\leq \sum_i \Pr[R \in A_x \oplus s_i] \\ &< m \cdot 2^{-n} < 1. \end{aligned}$$

In particular, there exists an $r \in \{0,1\}^m$ such that $r \notin \bigcup_i (A_x \oplus s_i)$.

$x \in L$: Choose $S_1, S_2, \ldots, S_m \stackrel{\text{R}}{\leftarrow} \{0,1\}^m$. Then, for every fixed r, we have:

$$\begin{aligned} \Pr[r \notin \bigcup_i (A_x \oplus S_i)] &= \prod_i \Pr[r \notin A_x \oplus S_i] \\ &= \prod_i \Pr[S_i \notin A_x \oplus r] \\ &< (2^{-n})^m, \end{aligned}$$

since A_x and hence $A_x \oplus r$ contains all but a 2^{-n} fraction of $\{0,1\}^m$. By a union bound, we have:

$$\Pr[\exists r \quad r \notin \bigcup_i (A_x \oplus S_i)] < 2^n \cdot (2^{-n})^m < 1.$$

Thus, there exist $s_1, \ldots, s_m$ such that $\bigcup_i (A_x \oplus s_i)$ contains all points r in $\{0,1\}^m$.

□

Readers familiar with complexity theory will recognize the above proof as showing that **BPP** is contained in the 2nd level of the *polynomial-time hierarchy* (**PH**). In general, the k'th level of the **PH** contains all languages that satisfy a k-quantifier expression analogous to (3.1).

3.4 The Method of Conditional Expectations

In the previous sections, we saw several derandomization techniques (enumeration, nonuniformity, nondeterminism) that are general in the sense that they apply to all of **BPP**, but are infeasible

in the sense that they cannot be implemented by efficient deterministic algorithms. In this section and the next one, we will see two derandomization techniques that sometimes can be implemented efficiently, but do not apply to all randomized algorithms.

3.4.1 The general approach

Consider a randomized algorithm that uses m random bits. We can view all its sequences of coin tosses as corresponding to a binary tree of depth m. We know that most paths (from the root to the leaf) are "good," i.e., give a correct answer. A natural idea is to try and find such a path by walking down from the root and making "good" choices at each step. Equivalently, we try to find a good sequence of coin tosses "bit-by-bit".

To make this precise, fix a randomized algorithm A and an input x, and let m be the number of random bits used by A on input x. For $1 \leq i \leq m$ and $r_1, r_2, \ldots, r_i \in \{0,1\}$, define $P(r_1, r_2, \ldots, r_i)$ to be the fraction of continuations that are good sequences of coin tosses. More precisely, if $R_1, \ldots, R_m$ are uniform and independent random bits, then

$$\begin{aligned} P(r_1, r_2, \ldots, r_i) &\stackrel{\text{def}}{=} \Pr_{R_1, R_2, \ldots, R_m}[A(x; R_1, R_2, \ldots, R_m) \text{ is correct } | R_1 = r_1, R_2 = r_2, \ldots, R_i = r_i] \\ &= \mathop{\mathrm{E}}_{R_{i+1}}[P(r_1, r_2, \ldots, r_i, R_{i+1})]. \end{aligned}$$

(See Figure 3.1.)

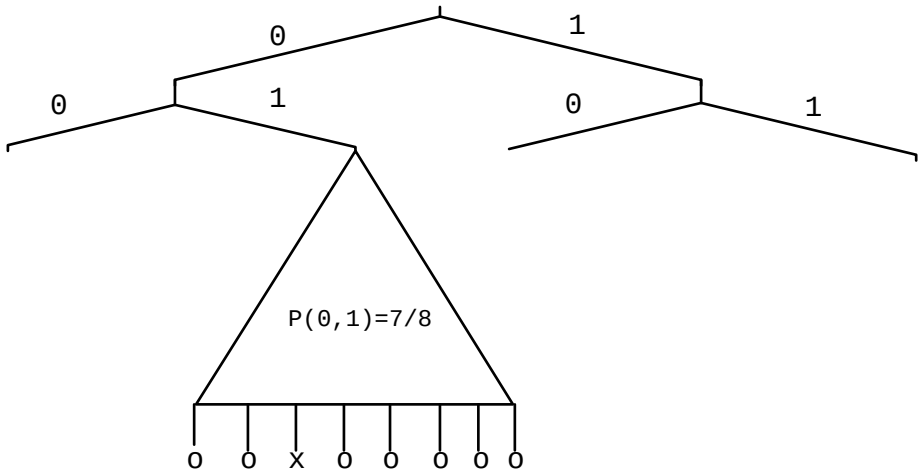

Fig. 3.1 An example of $P(r_1, r_2)$, where "o" at the leaf denotes a good path.

By averaging, there exists an $r_{i+1} \in \{0,1\}$ such that $P(r_1, r_2, \ldots, r_i, r_{i+1}) \geq P(r_1, r_2, \ldots, r_i)$. So at node $(r_1, r_2, \ldots, r_i)$, we simply pick r_{i+1} that maximizes $P(r_1, r_2, \ldots, r_i, r_{i+1})$. At the end we have $r_1, r_2, \ldots, r_m$, and

$$P(r_1, r_2, \ldots, r_m) \geq P(r_1, r_2, \ldots, r_{m-1}) \geq \cdots \geq P(r_1) \geq P(\Lambda) \geq 2/3$$

where $P(\Lambda)$ denotes the fraction of good paths from the root. Then $P(r_1, r_2, \ldots, r_m) = 1$, since it is either 1 or 0.

Note that to implement this method, we need to compute $P(r_1, r_2, \ldots, r_i)$ deterministically, and this may be infeasible. However, there are nontrivial algorithms where this method does work, often for *search* problems rather than decision problems, and where we measure not a boolean outcome (e.g. whether A is correct as above) but some other measure of quality of the output. Below we see one such example, where it turns out to yield a natural "greedy algorithm".

3.4.2 Derandomized MaxCut Approximation

Recall the MaxCut problem:

Computational Problem 3.15 (Computational Problem 2.38, rephrased).
MaxCut: Given a graph $G = (V, E)$, find a partition S, T of V (i.e. $S \cup T = V$, $S \cap T = \emptyset$) maximizing the size of the set $\text{cut}(S, T) = \{\{u, v\} \in E : u \in S, v \in T\}$.

We saw a simple randomized algorithm that finds a cut of (expected) size at least $|E|/2$ (not counting any self-loops, which can never be cut), which we now phrase in a way suitable for derandomization.

Algorithm 3.16 (randomized MaxCut, rephrased).
Input: a graph $G = ([N], E)$ (with no self-loops)

Flip N coins $r_1, r_2, \ldots, r_N$, put vertex i in S if $r_i = 0$ and in T if $r_i = 1$. Output (S, T).

To derandomize this algorithm using the Method of Conditional Expectations, define the conditional expectation

$$e(r_1, r_2, \ldots, r_i) \stackrel{\text{def}}{=} \mathop{\mathrm{E}}_{R_1, R_2, \ldots, R_N} \left[|\text{cut}(S, T)| \Big| R_1 = r_1, R_2 = r_2, \ldots, R_i = r_i \right]$$

to be the expected cut size when the random choices for the first i coins are fixed to $r_1, r_2, \ldots, r_i$.

We know that when no random bits are fixed, $e[\Lambda] \geq |E|/2$ (because each edge is cut with probability $1/2$), and all we need to calculate is $e(r_1, r_2, \ldots, r_i)$ for $1 \leq i \leq N$. For this particular algorithm it turns out that the quantity is not hard to compute. Let $S_i \stackrel{\text{def}}{=} \{j : j \leq i, r_j = 0\}$ (resp. $T_i \stackrel{\text{def}}{=} \{j : j \leq i, r_j = 1\}$) be the set of vertices in S (resp. T) after we determine $r_1, \ldots, r_i$, and $U_i \stackrel{\text{def}}{=} \{i+1, i+2, \ldots, N\}$ be the "undecided" vertices that have not been put into S or T. Then

$$e(r_1, r_2, \ldots, r_i) = |\text{cut}(S_i, T_i)| + 1/2 \left(|\text{cut}(U_i, [N])| \right). \tag{3.2}$$

Note that $\text{cut}(U_i, [N])$ is the set of *unordered* edges that have at least one endpoint in U_i. Now we can deterministically select a value for r_{i+1}, by computing and comparing $e(r_1, r_2, \ldots, r_i, 0)$ and $e(r_1, r_2, \ldots, r_i, 1)$.

In fact, the decision on r_{i+1} can be made even simpler than computing (3.2) in its entirety, by observing that the set $\text{cut}(U_{i+1}, [N])$ is independent of the choice of r_{i+1}. Therefore, to maximize $e(r_1, r_2 \ldots, r_i, r_{i+1})$, it is enough to choose r_{i+1} that maximizes the $|\text{cut}(S, T)|$ term. This term increases by either $|\text{cut}(\{i+1\}, T_i)|$ or $|\text{cut}(\{i+1\}, S_i)|$ depending on whether we place vertex $i+1$ in S or T, respectively. To summarize, we have

$$e(r_1, \ldots, r_i, 0) - e(r_1, \ldots, r_i, 1) = |\text{cut}(\{i+1\}, T_i)| - |\text{cut}(\{i+1\}, S_i)|.$$

This gives rise to the following deterministic algorithm, which is guaranteed to always find a cut of size at least $|E|/2$:

Algorithm 3.17 (deterministic MaxCut approximation).
Input: a graph $G = ([N], E)$ (with no self-loops)

(1) Set $S = \emptyset$, $T = \emptyset$
(2) For $i = 0, \ldots, N-1$:
  (a) If $|\text{cut}(\{i+1\}, T)| > |\text{cut}(\{i+1\}, S)|$, set $S \leftarrow S \cup \{i+1\}$,
  (b) Else set $T \leftarrow T \cup \{i+1\}$.

Note that this is the natural "greedy" algorithm for this problem. In other cases, the Method of Conditional Expectations yields algorithms that, while still arguably "greedy," would have been much less easy to find directly. Thus, designing a randomized algorithm and then trying to derandomize it can be a useful paradigm for the design of deterministic algorithms even if the randomization does not provide gains in efficiency.

3.5 Pairwise Independence

3.5.1 An Example

As a motivating example for pairwise independence, we give another way of derandomizing the MaxCut approximation algorithm discussed above. Recall the analysis of the randomized algorithm:

$$\mathrm{E}[|\text{cut}(S)|] = \sum_{(i,j)\in E} \Pr[R_i \neq R_j] = |E|/2,$$

where $R_1, \ldots, R_N$ are the random bits of the algorithm. The key observation is that this analysis applies for any distribution on $(R_1, \ldots, R_N)$ satisfying $\Pr[R_i \neq R_j] = 1/2$ for each $i \neq j$. Thus, they do not need to be completely independent random variables; it suffices for them to be *pairwise independent*. That is, each R_i is an unbiased random bit, and for each $i \neq j$, R_i is independent from R_j.

This leads to the question: Can we generate N pairwise independent bits using less than N truly random bits? The answer turns out to be *yes*, as illustrated by the following construction.

Construction 3.18 (pairwise independent bits). Let $B_1, \ldots, B_k$ be k independent unbiased random bits. For each nonempty $S \subset [k]$, let R_S be the random variable $\oplus_{i \in S} B_i$.

Proposition 3.19. The $2^k - 1$ random variables R_S in Construction 3.18 are pairwise independent unbiased random bits.

Proof. It is evident that each R_S is unbiased. For pairwise independence, consider any two nonempty sets $S \neq T \subset [k]$. Then:

$$\begin{aligned} R_S &= R_{S\cap T} \oplus R_{S\setminus T} \\ R_T &= R_{S\cap T} \oplus R_{T\setminus S}. \end{aligned}$$

Note that $R_{S\cap T}$, $R_{S\setminus T}$ and $R_{T\setminus S}$ are independent as they depend on disjoint subsets of the B_i's, and at least two of these subsets are nonempty. This implies that (R_S, R_T) takes each value in $\{0,1\}^2$ with probability $1/4$. □

Note that this gives us a way to generate N pairwise independent bits from $\lceil \log(N+1) \rceil$ independent random bits. Thus, we can reduce the randomness required by the MaxCut algorithm to logarithmic, and then we can obtain a deterministic algorithm by enumeration.

Algorithm 3.20 (deterministic MaxCut algorithm II).
Input: a graph $G = ([N], E)$ (with no self-loops)

For *all* sequences of bits $b_1, b_2, \ldots, b_{\lceil \log(N+1) \rceil}$, run the randomized MaxCut algorithm using coin tosses $(r_S = \oplus_{i\in S} b_i)_{S\neq\emptyset}$ and choose the largest cut thus obtained.

Since there are at most $2(N+1)$ sequences of b_i's, the derandomized algorithm still runs in polynomial time. It is slower than the algorithm we obtained by the Method of Conditional Expectations (Algorithm 3.17), but it has the advantage of using logarithmic workspace and being parallelizable. The derandomization can be sped up using almost pairwise independence (at the price of a slightly worse approximation factor); see Problem 3.4.

3.5.2 Pairwise Independent Hash Functions

Some applications require pairwise independent random variables that take values from a larger range, e.g. we want $N = 2^n$ pairwise independent random variables, each of which is uniformly distributed in $\{0,1\}^m = [M]$. (Throughout this survey, we will often use the convention that a lowercase letter equals the logarithm (base 2) of the corresponding capital letter.) The naive approach is to repeat the above algorithm for the individual bits m times. This uses roughly $n \cdot m = (\log M)(\log N)$ initial random bits, which is no longer logarithmic in N if M is nonconstant. Below we will see that we can do much better. But first some definitions.

A sequence of N random variables each taking a value in $[M]$ can be viewed as a distribution on sequences in $[M]^N$. Another interpretation of such a sequence is as a mapping $f : [N] \to [M]$. The latter interpretation turns out to be more useful when discussing the computational complexity of the constructions.

Definition 3.21 (Pairwise Independent Hash Functions). A family (i.e. multiset) of functions $\mathcal{H} = \{h : [N] \to [M]\}$ is *pairwise independent* if the following two conditions hold when $H \overset{R}{\leftarrow} \mathcal{H}$ is a function chosen uniformly at random from $\mathcal{H}$:

(1) $\forall x \in [N]$, the random variable $H(x)$ is uniformly distributed in $[M]$.

(2) $\forall x_1 \neq x_2 \in [N]$, the random variables $H(x_1)$ and $H(x_2)$ are independent.

Equivalently, we can combine the two conditions and require that

$$\forall x_1 \neq x_2 \in [N], \forall y_1, y_2 \in [M], \Pr_{H \xleftarrow{R} \mathcal{H}}[H(x_1) = y_1 \wedge H(x_2) = y_2] = \frac{1}{M^2}.$$

Note that the probability above is over the random choice of a function from the family $\mathcal{H}$. This is why we talk about a family of functions rather than a single function. The description in terms of functions makes it natural to impose a strong efficiency requirement:

Definition 3.22. A family of functions $\mathcal{H} = \{h : [N] \to [M]\}$ is *explicit* if given the description of h and $x \in [N]$, the value $h(x)$ can be computed in time $\text{poly}(\log N, \log M)$.

Pairwise independent hash functions are sometimes referred to as *strongly 2-universal hash functions*, to contrast with the weaker notion of *2-universal hash functions*, which requires only that $\Pr[H(x_1) = H(x_2)] \leq 1/M$ for all $x_1 \neq x_2$. (Note that this property is all we needed for the deterministic MaxCut algorithm, and it allows for a small savings in that we can also include $S = \emptyset$ in Construction 3.18.)

Below we present another construction of a pairwise independent family.

Construction 3.23 (pairwise independent hash functions from linear maps). Let $\mathbb{F}$ be a finite field. Define the family of functions $\mathcal{H} = \{\ h_{a,b} : \mathbb{F} \to \mathbb{F}\}_{a,b \in \mathbb{F}}$ where $h_{a,b}(x) = ax + b$.

Proposition 3.24. The family of functions $\mathcal{H}$ in Construction 3.23 is pairwise independent.

Proof. Notice that the graph of the function $h_{a,b}(x)$ is the line with slope a and y-intercept b. Given $x_1 \neq x_2$ and y_1, y_2, there is exactly one such line containing the points (x_1, y_1) and (x_2, y_2) (namely, the line with slope $a = (y_1 - y_2)/(x_1 - x_2)$ and y-intercept $b = y_1 - ax_1$). Thus, the probability over a, b that $h_{a,b}(x_1) = y_1$ and $h_{a,b}(x_2) = y_2$ equals the reciprocal of the number of lines, namely $1/|\mathbb{F}|^2$. □

This construction uses $2 \log |\mathbb{F}|$ random bits, since we have to choose a and b at random from $\mathbb{F}$ to get a function $h_{a,b} \xleftarrow{R} \mathcal{H}$. Compare this to $|\mathbb{F}| \log |\mathbb{F}|$ bits required to choose a truly random function, and $(\log |\mathbb{F}|)^2$ bits for repeating the construction of Proposition 3.19 for each output bit.

Note that evaluating the functions of Construction 3.23 requires a description of the field $\mathbb{F}$ that enables us to perform addition and multiplication of field elements. Thus we take a brief aside to review the complexity of constructing and computing in finite fields.

Remark 3.25 (constructing finite fields). Recall that for every prime power $q = p^k$ there is a field $\mathbb{F}_q$ (often denoted $\text{GF}(q)$) of size q, and this field is unique up to isomorphism (renaming elements). The prime p is called the *characteristic* of the field. $\mathbb{F}_q$ has a description of length $O(\log q)$ enabling addition, multiplication, and division to be performed in polynomial time (i.e.

time $\text{poly}(\log q)$). (This description is simply an irreducible polynomial f of degree k over $\mathbb{F}_p = \mathbb{Z}_p$.) To construct this description, we first need to determine the characteristic p; finding a prime p of a desired bitlength ℓ can be done probabilistically in time $\text{poly}(\ell) = \text{poly}(\log p)$ and deterministically in time $\text{poly}(2^\ell) = \text{poly}(p)$. Then given p and k, a description of $\mathbb{F}_q$ (for $q = p^k$) can be found probabilistically in time $\text{poly}(\log p, k) = \text{poly}(\log q)$ and deterministically in time $\text{poly}(p, k)$. Note that for both steps, the deterministic algorithms are exponential in the bitlength of the characteristic p. Thus, for computational purposes, a convenient choice is often to instead take $p = 2$ and k large, in which case everything can be done deterministically in time $\text{poly}(k) = \text{poly}(\log q)$.

Using a finite fields of size 2^n as suggested above, we obtain an explicit construction of pairwise independent hash functions $\mathcal{H}_{n,n} = \{h : \{0,1\}^n \to \{0,1\}^n\}$ for every n. What if we want a family $\mathcal{H}_{n,m}$ of pairwise independent hash functions where the input length n and output length m are not equal? For $n < m$, we can take hash functions h from $\mathcal{H}_{m,m}$ and restrict their domain to $\{0,1\}^n$ by defining $h'(x) = h(x \circ 0^{m-n})$. In the case that $m < n$, we can take h from $\mathcal{H}_{n,n}$ and throw away $n - m$ bits of the output. That is, define $h'(x) = h(x)|_m$, where $h(x)|_m$ denotes the first m bits of $h(x)$.

In both cases, we use $2\max\{n, m\}$ random bits. This is the best possible when $m \geq n$. When $m < n$, it can be reduced to $m + n$ random bits by using $(ax)|_m + b$ where $b \in \{0,1\}^m$ instead of $(ax + b)|_m$. Summarizing:

Theorem 3.26. For every $n, m \in \mathbb{N}$, there is an explicit family of pairwise independent functions $\mathcal{H}_{n,m} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ where a random function from $\mathcal{H}_{n,m}$ can be selected using $\max\{m, n\} + m$ random bits.

Problem 3.5 shows that $\max\{m, n\} + m$ random bits is essentially optimal.

3.5.3 Hash Tables

The original motivating application for pairwise independent functions was for hash tables. Suppose we want to store a set $S \subset [N]$ of values and answer queries of the form "Is $x \in S$?" efficiently (or, more generally, acquire some piece of data associated with key x in case $x \in S$). A simple solution is to have a table T such that $T[x] = 1$ if and only if $x \in S$. But this requires N bits of storage, which is inefficient if $|S| \ll N$.

A better solution is to use hashing. Assume that we have a hash function $h : [N] \to [M]$ for some M to be determined later. Let T be a table of size M, initially empty. For each $x \in [N]$, we let $T[h(x)] = x$ if $x \in S$. So to test whether a given $y \in S$, we compute $h(y)$ and check if $T[h(y)] = y$. In order for this construction to be well-defined, we need h to be one-to-one on the set S. Suppose we choose a random function H from $[N]$ to $[M]$. Then, for any set S of size at most K, the probability that there are any collisions is

$$\Pr[\exists\ x \neq y \text{ s.t. } H(x) = H(y)] \leq \sum_{x \neq y \in S} \Pr[H(x) = H(y)] = \binom{K}{2} \cdot \frac{1}{M} < \varepsilon$$

for $M = K^2/\varepsilon$. Notice that the above analysis does not require H to be a completely random function; it suffices that H be pairwise independent (or even 2-universal). Thus using Theorem 3.26,

we can generate and store H using $O(\log N)$ random bits. The storage required for the table T is $O(M \log N) = O(K^2 \log N)$ bits, which is much smaller than N when $K = N^{o(1)}$. Note that to uniquely represent a set of size K, we need space at least $\log \binom{N}{K} = \Omega(K \log N)$ (when $K \leq N^{.99}$). In fact, there is a data structure achieving a matching space upper bound of $O(K \log N)$, which works by taking $M = O(K)$ in the above construction and using additional hash functions to separate the (few) collisions that will occur.

Often, when people analyze applications of hashing in computer science, they model the hash function as a truly random function. However, the domain of the hash function is often exponentially large, and thus it is infeasible to even write down a truly random hash function. Thus, it would be preferable to show that some explicit family of hash function works for the application with similar performance. In many cases (such as the one above), it can be shown that pairwise independence (or k-wise independence, as discussed below) suffices.

3.5.4 Randomness-Efficient Error Reduction and Sampling

Suppose we have a **BPP** algorithm for a language L that has a constant error probability. We want to reduce the error to 2^{-k}. We have already seen that this can be done using $O(k)$ independent repetitions (by a Chernoff Bound). If the algorithm originally used m random bits, then we use $O(km)$ random bits after error reduction. Here we will see how to reduce the number of random bits required for error reduction by doing repetitions that are only pairwise independent.

To analyze this, we will need an analogue of the Chernoff Bound (Theorem 2.21) that applies to averages of pairwise independent random variables. This will follow from Chebyshev's Inequality, which bounds the deviations of a random variable X from its mean μ in terms its *variance* $\mathrm{Var}[X] = \mathrm{E}[(X-\mu)^2] = \mathrm{E}[X^2] - \mu^2$.

Lemma 3.27 (Chebyshev's Inequality). If X is a random variable with expectation μ, then

$$\Pr[|X - \mu| \geq \varepsilon] \leq \frac{\mathrm{Var}[X]}{\varepsilon^2}$$

Proof. Applying Markov's Inequality (Lemma 2.20) to the random variable $Y = (X-\mu)^2$, we have:

$$\Pr[|X - \mu| \geq \varepsilon] = \Pr\left[(X-\mu)^2 \geq \varepsilon^2\right] \leq \frac{\mathrm{E}\left[(X-\mu)^2\right]}{\varepsilon^2} = \frac{\mathrm{Var}[X]}{\varepsilon^2}.$$

□

We now use this to show that a sum of pairwise independent random variables is concentrated around its expectation.

Proposition 3.28 (Pairwise Independent Tail Inequality). Let $X_1, \ldots, X_t$ be pairwise independent random variables taking values in the interval $[0,1]$, let $X = (\sum_i X_i)/t$, and $\mu = \mathrm{E}[X]$. Then

$$\Pr[|X - \mu| \geq \varepsilon] \leq \frac{1}{t\varepsilon^2}.$$

Proof. Let $\mu_i = \mathrm{E}[X_i]$. Then

$$\begin{aligned}
\mathrm{Var}[X] &= \mathrm{E}\left[(X-\mu)^2\right] \\
&= \frac{1}{t^2} \cdot \mathrm{E}\left[\left(\sum_i (X_i - \mu_i)\right)^2\right] \\
&= \frac{1}{t^2} \cdot \sum_{i,j} \mathrm{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right] \\
&= \frac{1}{t^2} \cdot \sum_i \mathrm{E}\left[(X_i - \mu_i)^2\right] \qquad \text{(by pairwise independence)} \\
&= \frac{1}{t^2} \cdot \sum_i \mathrm{Var}\left[X_i\right] \\
&\leq \frac{1}{t}
\end{aligned}$$

Now apply Chebyshev's Inequality. □

While this requires less independence than the Chernoff Bound, notice that the error probability decreases only linearly rather than exponentially with the number t of samples.

Error Reduction. Proposition 3.28 tells us that if we use $t = O(2^k)$ pairwise independent repetitions, we can reduce the error probability of a **BPP** algorithm from $1/3$ to 2^{-k}. If the original **BPP** algorithm uses m random bits, then we can do this by choosing $h : \{0,1\}^{k+O(1)} \to \{0,1\}^m$ at random from a pairwise independent family, and running the algorithm using coin tosses $h(x)$ for all $x \in \{0,1\}^{k+O(1)}$. This requires $m + \max\{m, k + O(1)\} = O(m+k)$ random bits.

	Number of Repetitions	Number of Random Bits
Independent Repetitions	$O(k)$	$O(km)$
Pairwise Independent Repetitions	$O(2^k)$	$O(m+k)$

Note that we saved substantially on the number of random bits, but paid a lot in the number of repetitions needed. To maintain a polynomial-time algorithm, we can only afford $k = O(\log n)$. This setting implies that if we have a **BPP** algorithm with constant error that uses m random bits, we have another **BPP** algorithm that uses $O(m + \log n) = O(m)$ random bits and has error $1/\mathrm{poly}(n)$. That is, we can go from constant to inverse-polynomial error only paying a constant factor in randomness. (In fact, it turns out there is a way to achieve this with *no* penalty in randomness; see Problem 4.6.)

Sampling. Recall the SAMPLING problem: Given oracle access to a function $f : \{0,1\}^m \to [0,1]$, we want to approximate $\mu(f)$ to within an additive error of ε.

In Section 2.3.1, we saw that we can solve this problem with probability $1-\delta$ by outputting the average of f on a random sample of $t = O(\log(1/\delta)/\varepsilon^2)$ points in $\{0,1\}^m$, where the correctness follows from the Chernoff Bound. To reduce the number of truly random bits used, we can use a pairwise independent sample instead. Specifically, taking $t = 1/(\varepsilon^2\delta)$ pairwise independent points, we get an error probability of at most δ (by Proposition 3.28). To generate t pairwise independent

samples of m bits each, we need $O(m + \log t) = O(m + \log(1/\varepsilon) + \log(1/\delta))$ truly random bits.

	Number of Samples	Number of Random Bits
Truly Random Sample	$O((1/\varepsilon^2) \cdot \log(1/\delta))$	$O(m \cdot (1/\varepsilon^2) \cdot \log(1/\delta))$
Pairwise Independent Repetitions	$O((1/\varepsilon^2) \cdot (1/\delta))$	$O(m + \log(1/\varepsilon) + \log(1/\delta))$

Both of these sampling algorithms have a natural restricted structure. First, they choose all of their queries to the oracle f nonadaptively, based solely on their coin tosses and not based on the answers to previous queries. Second, their output is simply the average of the queried values, whereas the original sampling problem does not constrain the output function. It is useful to abstract these properties as follows.

Definition 3.29. A *sampler* Samp for domain size M is given "coin tosses" $x \xleftarrow{R} [N]$ and outputs a sequence of samples $z_1, \ldots, z_t \in [M]$. We say that $\mathrm{Samp} : [N] \to [M]^t$ is a (δ, ε) *averaging sampler* if for every function $f : [M] \to [0, 1]$, we have

$$\Pr_{(z_1,\ldots,z_t) \xleftarrow{R} \mathrm{Samp}(U_{[N]})} \left[\frac{1}{t} \sum_{i=1}^{t} f(z_i) > \mu(f) + \varepsilon \right] \leq \delta. \tag{3.3}$$

If Inequality 3.3 only holds for f with (boolean) range $\{0, 1\}$, we call Samp a *boolean averaging sampler*. We say that Samp is *explicit* if given $x \in [N]$ and $i \in [t]$, $\mathrm{Samp}(x)_i$ can be computed in time $\mathrm{poly}(\log N, \log t)$.

We note that, in contrast to the Chernoff Bound (Theorem 2.21) and the Pairwise Independent Tail Inequality (Proposition 3.28), this definition seems to only provide an error guarantee in one direction, namely that the sample average does not significantly *exceed* the global average (except with small probability). However, a guarantee in the other direction also follows by considering the function $1 - f$. Thus, up to a factor of 2 in the failure probability δ, the above definition is equivalent to requiring that $\Pr[|(1/t) \cdot \sum_i f(z_i) - \mu(f)| > \varepsilon] \leq \delta$. We choose to use a one-sided guarantee because it will make the connection to list-decodable codes (in Chapter 5) slightly cleaner.

Our pairwise-independent sampling algorithm can now be described as follows:

Theorem 3.30 (Pairwise Independent Sampler). For every $m \in \mathbb{N}$ and $\delta, \varepsilon \in [0, 1]$, there is an explicit (δ, ε) averaging sampler $\mathrm{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ using $n = O(m + \log(1/\varepsilon) + \log(1/\delta))$ random bits and $t = O(1/(\varepsilon^2 \delta))$ samples.

As we will see in Volume II, averaging samplers are intimately related to the other pseudorandom objects we are studying. In addition, some applications of samplers require samplers of this restricted form.

3.5.5 k-wise Independence

Our definition and construction of pairwise independent functions generalize naturally to k-wise independence for any k.

Definition 3.31 (k-wise independent hash functions). For $N, M, k \in \mathbb{N}$ such that $k \leq N$, a family of functions $\mathcal{H} = \{h : [N] \to [M]\}$ is *k-wise independent* if for all distinct $x_1, x_2, \ldots, x_k \in [N]$, the random variables $H(x_1), \ldots, H(x_k)$ are independent and uniformly distributed in $[M]$ when $H \xleftarrow{R} \mathcal{H}$.

Construction 3.32 (k-wise independence from polynomials). Let $\mathbb{F}$ be a finite field. Define the family of functions $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_{k-1}} : \mathbb{F} \to \mathbb{F}\}$ where each $h_{a_0,a_1,\ldots,a_{k-1}}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1}x^{k-1}$ for $a, b \in \mathbb{F}$.

Proposition 3.33. The family $\mathcal{H}$ given in Construction 3.32 is k-wise independent.

Proof. Similarly to the proof of Proposition 3.24, it suffices to prove that for all distinct $x_1, \ldots, x_k \in \mathbb{F}$ and all $y_1, \ldots, y_k \in \mathbb{F}$, there is exactly one polynomial h of degree at most $k-1$ such that $h(x_i) = y_i$ for all i. To show that such a polynomial exists, we can use the Lagrange Interpolation Formula:

$$h(x) = \sum_{i=1}^{k} y_i \cdot \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

To show uniqueness, suppose we have two polynomials h and g of degree at most $k-1$ such that $h(x_i) = g(x_i)$ for $i = 1, \ldots, k$. Then $h - g$ has at least k roots, and thus must be the zero polynomial. □

Corollary 3.34. For every $n, m, k \in \mathbb{N}$, there is a family of k-wise independent functions $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ such that choosing a random function from $\mathcal{H}$ takes $k \cdot \max\{n, m\}$ random bits, and evaluating a function from $\mathcal{H}$ takes time $\mathrm{poly}(n, m, k)$.

k-wise independent hash functions have applications similar to those that pairwise independent hash functions have. The increased independence is crucial in derandomizing some algorithms. k-wise independent random variables also satisfy a tail bound similar to Proposition 3.28, with the key improvement being that the error probability vanishes linearly in $t^{k/2}$ rather than t; see Problem 3.8.

3.6 Exercises

~~**Problem 3.1 (Derandomizing RP versus BPP*).** Show that **prRP** = **prP** implies that **prBPP** = **prP**, and thus also that **BPP** = **P**. (Hint: Look at the proof that **NP** = **P** ⇒ **BPP** = **P**.)~~

Problem 3.2 (Designs). Designs (also known as packings) are collections of sets that are nearly disjoint. In Chapter 7, we will see how they are useful in the construction of pseudorandom generators. Formally, a collection of sets $S_1, S_2, \ldots, S_m \subset [d]$ is called an (ℓ, a)-*design* (for integers $a \leq \ell \leq d$) if

- For all i, $|S_i| = \ell$.
- For all $i \neq j$, $|S_i \cap S_j| < a$.

For given ℓ, we'd like m to be large, a to be small, and d to be small. That is, we'd like to pack many sets into a small universe with small intersections.

(1) Prove that if $m \leq \binom{d}{a} / \binom{\ell}{a}^2$, then there exists an (ℓ, a)-design $S_1, \ldots, S_m \subset [d]$.
Hint: Use the Probabilistic Method. Specifically, show that if the sets are chosen randomly, then for every $S_1, \ldots, S_{i-1}$,

$$\mathop{\mathrm{E}}_{S_i}\left[\#\{j < i : |S_i \cap S_j| \geq a\}\right] < 1.$$

(2) Conclude that for every constant $\gamma > 0$ and every $\ell, m \in \mathbb{N}$, there exists an (ℓ, a)-design $S_1, \cdots, S_m \subseteq [d]$ with $d = O\left(\frac{\ell^2}{a}\right)$ and $a = \gamma \cdot \log m$. In particular, setting $m = 2^\ell$, we fit exponentially many sets of size ℓ in a universe of size $d = O(\ell)$ while keeping the intersections an arbitrarily small fraction of the set size.

(3) Using the Method of Conditional Expectations, show how to construct designs as in Parts 1 and 2 *deterministically* in time $\mathrm{poly}(m, d)$.

Problem 3.3 (More Pairwise Independent Families). (1) (matrix-vector family) For an $n \times m$ $\{0,1\}$-matrix A and $b \in \{0,1\}^n$, define a function $h_{A,b} : \{0,1\}^m \to \{0,1\}^n$ by $h_{A,b}(x) = (Ax + b) \bmod 2$. (The "mod 2" is applied componentwise.) Show that $\mathcal{H}_{m,n} = \{h_{A,b}\}$ is a pairwise independent family. Compare the number of random bits needed to generate a random function in $\mathcal{H}_{m,n}$ to Construction 3.23.

(2) (Toeplitz matrices) A is a *Toeplitz matrix* if it is constant on diagonals, i.e. $A_{i+1,j+1} = A_{i,j}$ for all i, j. Show that even if we restrict the family $\mathcal{H}_{m,n}$ in Part 1 to only include $h_{A,b}$ for Toeplitz matrices A, we still get a pairwise independent family. How many random bits are needed now?

Problem 3.4 (Almost Pairwise Independence). A family of functions $\mathcal{H}$ mapping domain $[N]$ to range $[M]$ is ε-*almost pairwise independent*[3] if for every $x_1 \neq x_2 \in [N]$, $y_1, y_2 \in [M]$, we have

$$\Pr_{H \xleftarrow{R} \mathcal{H}}\left[H(x_1) = y_1 \text{ and } H(x_2) = y_2\right] \leq \frac{1+\varepsilon}{M^2}.$$

[3] Another common definition of ε-almost pairwise independence requires instead that for every $x_1 \neq x_2 \in [N]$, if we choose a random hash function $H \xleftarrow{R} \mathcal{H}$, the random variable $(H(x_1), H(x_2))$ is ε-close to two uniform and independent elements of $[M]$ in statistical difference (as defined in Chapter 6). The two definitions are equivalent up to a factor of M^2 in the error parameter ε.

(1) Show that there exists a family $\mathcal{H}$ of ε-almost pairwise independent functions from $\{0,1\}^n$ to $\{0,1\}^m$ such that choosing a random function from $\mathcal{H}$ requires only $O(m + \log n + \log(1/\varepsilon))$ random bits (as opposed to $O(m+n)$ for exact pairwise independence). (Hint: First consider domain $\mathbb{F}^{d+1}$ for an appropriately chosen finite field $\mathbb{F}$ and $d \in \mathbb{N}$, and look at maps of the form $h = g \circ f_a$, where g comes from some pairwise independent family and $f_a : \mathbb{F}^{d+1} \to \mathbb{F}$ is defined by $f_a(x_0, \ldots, x_d) = x_0 + x_1 a + x_2 a^2 + \cdots + x_d a^d$.)

(2) Give a deterministic algorithm that on input an N-vertex, M-edge graph G (with no self-loops), finds a cut of size at least $(1/2 - o(1)) \cdot M$ in time $M \cdot \mathrm{polylog}(N)$ and space $O(\log M)$ (thereby improving the $M \cdot \mathrm{poly}(N)$ running time of Algorithm 3.20).

Problem 3.5 (Size Lower Bound for Pairwise Independent Families). Let $\mathcal{H} = \{h : [N] \to [M]\}$ be a pairwise independent family of functions.

(1) Prove that if $N \geq 2$, then $|\mathcal{H}| \geq M^2$.

(2) Prove that if $M = 2$, then $|\mathcal{H}| \geq N + 1$. (Hint: based on $\mathcal{H}$, construct a sequence of orthogonal vectors $v_x \in \{\pm 1\}^{|\mathcal{H}|}$ parameterized by $x \in [N]$.)

(3) More generally, prove that for arbitrary M, we have $|\mathcal{H}| \geq N \cdot (M-1) + 1$. (Hint: for each $x \in [N]$, construct $M-1$ linearly independent vectors $v_{x,y} \in \mathbb{R}^{|\mathcal{H}|}$ such that $v_{x,y} \perp v_{x',y}$ if $x \neq x'$.)

(4) Deduce that for $N = 2^n$ and $M = 2^m$, selecting a random function from $\mathcal{H}$ requires at least $\max\{n, m\} + m$ random bits.

Problem 3.6 (Frequency Moments of Data Streams). Given one pass through a huge "stream" of data items $(a_1, a_2, \ldots, a_k)$, where each $a_i \in \{0,1\}^n$, we want to compute statistics on the distribution of items occurring in the stream while using small space (not enough to store all the items or maintain a histogram). In this problem, you will see how to compute the *2nd frequency moment* $f_2 = \sum_a m_a^2$, where $m_a = \#\{i : a_i = a\}$.

The algorithm works as follows: Before receiving any items, it chooses t random *4-wise independent* hash functions $H_1, \ldots, H_t : \{0,1\}^n \to \{+1, -1\}$, and sets counters $X_1 = X_2 = \cdots = X_t = 0$. Upon receiving the i'th item a_i, it adds $H_j(a_i)$ to counter X_j. At the end of the stream, it outputs $Y = (X_1^2 + \cdots + X_t^2)/t$.

Notice that the algorithm only needs space $O(t \cdot n)$ to store the hash functions H_j and space $O(t \cdot \log k)$ to maintain the counters X_j (compared to space $k \cdot n$ to store the entire stream, and space $2^n \cdot \log k$ to maintain a histogram).

(1) Show that for every data stream $(a_1, \ldots, a_k)$ and each j, we have $\mathrm{E}[X_j^2] = f_2$, where the expectation is over the choice of the hash function H_j.

(2) Show that $\mathrm{Var}[X_j^2] \leq 2 f_2^2$.

(3) Conclude that for a sufficiently large constant t (independent of n and k), the output Y is within 1% of f_2 with probability at least .99.

Problem 3.7 (Improved Pairwise Independent Tail Inequality). (1) Show that if X is a random variable taking values in $[0,1]$ and $\mu = \mathrm{E}[X]$, we have $\mathrm{Var}[X] \leq \mu \cdot (1-\mu)$.

(2) Improve the Pairwise Independent Tail Inequality (Proposition 3.28) to show that if X is the average of t pairwise independent random variables taking values in $[0,1]$ and $\mu = \mathrm{E}[X]$, then $\Pr[|X-\mu| \geq \varepsilon] \leq \mu \cdot (1-\mu)/(t \cdot \varepsilon^2)$. In particular, for $t = O(1/\mu)$, we have $\Pr[.99\mu \leq X \leq 1.01\mu] \geq .99$.

Problem 3.8 (k-wise Independent Tail Inequality). Let X be the average of t k-wise independent random variables for an even integer k, and let $\mu = \mathrm{E}[X]$. Prove that

$$\Pr[|X-\mu| \geq \varepsilon] \leq \left(\frac{k^2}{4t\varepsilon^2}\right)^{k/2}.$$

(Hint: show that all terms in the expansion of $\mathrm{E}[(X-\mu)^k] = \mathrm{E}[(\sum_i (X_i - \mu_i))^k]$ that involve more than $k/2$ variables X_i are zero.) Note that for fixed k, this probability decays like $O(1/(t\varepsilon^2))^{k/2}$, improving the $1/(t\varepsilon^2)$ bound in pairwise independence when $k > 2$.

Problem 3.9 (Hitting Samplers). A function $\mathrm{Samp} : [N] \to [M]^t$ is a (δ, ε) *hitting sampler* if for every set $T \subseteq [M]$ of density greater than ε, we have

$$\Pr_{(z_1,\ldots,z_t) \xleftarrow{R} \mathrm{Samp}(U_{[N]})} [\exists i \quad z_i \in T] \geq 1 - \delta.$$

(1) Show that every (δ, ε) averaging sampler is a (δ, ε) hitting sampler.

(2) Show that if we only want a hitting sampler, the number of samples in Theorem 3.30 can be reduced to $O(1/(\varepsilon\delta))$. (Hint: use Problem 3.7.)

(3) For which subset of **BPP** algorithms are hitting samplers useful for doing randomness-efficient error reduction?

3.7 Chapter Notes and References

The Time Hierarchy Theorem was proven by Hartmanis and Stearns [HS]; proofs can be found in any standard text on complexity theory, e.g. [Sip3, Gol6, AB]. Adleman [Adl] showed that every language in **RP** has polynomial-sized circuits (cf., Corollary 3.12), and this was generalized to **BPP** by Gill. Pippenger [Pip] showed the equivalence between having polynomial-sized circuits and **P/poly** (Fact 3.11). The general definition of complexity classes with advice (Definition 3.10) is due to Karp and Lipton [KL], who explored the relationship between nonuniform lower bounds and uniform lower bounds. A $5n - o(n)$ circuit-size lower bound for an explicit function (in **P**) was given by Iwama et al. [LR, IM].

The existence of universal traversal sequences (Example 3.8) was proven by Aleliunas et al. [AKL$^+$], who suggested finding an explicit construction (Open Problem 3.9) as an approach

to derandomizing the logspace algorithm for UNDIRECTED S-T CONNECTIVITY. For the state of the art on these problems, see Section 4.4.

Theorem 3.14 is due to Sipser [Sip1], who proved that **BPP** is in the 4th level of the polynomial-time hierarchy; this was improved to the 2nd level by Gács. Our proof of Theorem 3.14 is due to Lautemann [Lau]. Problem 3.1 is due to Buhrman and Fortnow [BF]. For more on nondeterministic computation and nonuniform complexity, see textbooks on computational complexity, such as [Sip3, Gol6, AB].

The Method of Conditional Probabilities was formalized and popularized as an algorithmic tool in the work of Spencer [Spe] and Raghavan [Rag]. Its use in Algorithm 3.17 for approximating MAXCUT is implicit in [Lub2]. For more on this method, see the textbooks [MR, AS1].

A more detailed treatment of pairwise independence (along with a variety of other topics in pseudorandomness and derandomization) can be found in the survey by Luby and Wigderson [LW]. The use of limited independence in computer science originates with the seminal papers of Carter and Wegman [CW, WC], which introduced the notions of universal, strongly universal (i.e. k-wise independent), and almost strongly universal (i.e. almost k-wise independent) families of hash functions. The pairwise independent and k-wise independent sample spaces of Constructions 3.18, 3.23, and 3.32 date back to the work of Lancaster [Lan] and Joffe [Jof1, Jof2] in the probability literature, and were rediscovered several times in the computer science literature. The construction of pairwise independent hash functions from Part 1 of Problem 3.3 is due to Carter and Wegman [CW] and Part 2 is implicit in [Vaz2, GL]. The application to hash tables from Section 3.5.3 is due to Carter and Wegman [CW], and the method mentioned for improving the space complexity to $O(K \log N)$ is due to Fredman, Komlós, and Szemerédi [FKS]. The problem of randomness-efficient error reduction (sometimes called "deterministic amplification") was first studied by Karp, Pippenger, and Sipser [KPS], and the method using pairwise independence given in Section 3.5.4 was proposed by Chor and Goldreich [CG1]. The use of pairwise independence for derandomizing algorithms was pioneered by Luby [Lub1]; Algorithm 3.20 for MAXCUT is implicit in [Lub2]. The notion of averaging samplers was introduced by Bellare and Rompel [BR] (under the term "oblivious samplers"). For more on samplers and averaging samplers, see the survey by Goldreich [Gol1]. Tail bounds for k-wise independent random variables, such as the one in Problem 3.8, can be found in the papers [CG1, BR, SSS].

Problem 3.2 on designs is from [EFF], with the derandomization of Part 3 being from [NW, LW]. Problem 3.6 on the frequency moments of data streams is due to Alon, Matias, and Szegedy [AMS]. For more on data stream algorithms, we refer to the survey by Muthukrishnan [Mut].

4

Expander Graphs

Now that we have seen a variety of basic derandomization techniques, we will move on to study the first major "pseudorandom object" in this survey, *expander graphs*. These are graphs that are "sparse" yet very "well-connected."

4.1 Measures of Expansion

We will typically interpret the properties of expander graphs in an asymptotic sense. That is, there will be an infinite family of graphs G_i, with a growing number of vertices N_i. By "sparse," we mean that the degree D_i of G_i should be very slowly growing as a function of N_i. The "well-connectedness" property has a variety of different interpretations, which we will discuss below. Typically, we will drop the subscripts of i and the fact that we are talking about an infinite family of graphs will be implicit in our theorems. As in Section 2.4.2, we will state many of our definitions for *directed multigraphs* (which we'll call *digraphs* for short), though in the end we will mostly study undirected multigraphs.

4.1.1 Vertex Expansion

The classic measure of well-connectedness in expanders requires that every "not-too-large" set of vertices has "many" neighbors:

Definition 4.1. A digraph G is a (K, A) *vertex expander* if for all sets S of at most K vertices, the *neighborhood* $N(S) \stackrel{\text{def}}{=} \{u | \exists v \in S \text{ s.t. } (u, v) \in E\}$ is of size at least $A \cdot |S|$.

Ideally, we would like $D = O(1)$, $K = \Omega(N)$ where N is the number of vertices, and A as close to D as possible.

There are several other measures of expansion, some of which we will examine in forthcoming sections:

- Boundary expansion: instead of $N(S)$, only use the *boundary* $\partial S \stackrel{\text{def}}{=} N(S) \setminus S$.

- Edge expansion (cuts): instead of ∂S, use the number of edges leaving S.
- Random walks: random walks converge quickly to the uniform distribution, i.e. the second eigenvalue $\lambda(G)$ is small.
- "Quasi-randomness" (a.k.a "Mixing"): for every two sets S and T (say of constant density), the fraction of edges between S and T is roughly the product of their densities.

All of these measures are very closely related to each other, and are even equivalent for certain settings of parameters.

It is not obvious from the definition that good vertex expanders (say, with $D = O(1)$, $K = \Omega(N)$, and $A = 1 + \Omega(1)$) even exist. We will show this using the Probabilistic Method.

Theorem 4.2. For all constants $D \geq 3$, there is a constant $\alpha > 0$ such that for all sufficiently large N, a random D-regular undirected graph on N vertices is an $(\alpha N, D - 1.01)$ vertex expander with probability at least $1/2$.

Note that the degree bound of 3 is the smallest possible, as every graph of degree 2 is a poor expander (being a union of cycles and chains). In addition, for most settings of parameters, it is impossible to have expansion larger than $D - 1$ (as shown in Problem 4.3).

We prove a slightly simpler theorem for *bipartite expanders*.

Definition 4.3. A bipartite multigraph G is a (K, A) *vertex expander* if for all sets S of *left*-vertices of size at most K, the neighborhood $N(S)$ is of size at least $A \cdot |S|$.

Now, let $\mathrm{Bip}_{N,D}$ be the set of bipartite multigraphs that have N vertices on each side and are *D-leftregular*, meaning that every vertex on the left has D neighbors, numbered from $1, \ldots, D$ (but vertices on the right may have varying degrees).

Theorem 4.4. For every constant D, there exists a constant $\alpha > 0$ such that for all sufficiently large N, a uniformly random graph from $\mathrm{Bip}_{N,D}$ is an $(\alpha N, D-2)$ vertex expander with probability at least $1/2$.

Proof. First, note that choosing $G \xleftarrow{\mathrm{R}} \mathrm{Bip}_{N,D}$ is equivalent to uniformly and independently choosing D neighbors on the right for each left vertex v. Now, for $K \leq \alpha N$, let p_K be the probability that there exists a left-set S of size exactly K that does not expand by at least $D-2$. Fixing a subset S of size K, $N(S)$ is a set of KD random vertices in $[N]$ (chosen with replacement). We can imagine these vertices $V_1, V_2, \ldots, V_{KD}$ being chosen in sequence. Call V_i a *repeat* if $V_i \in \{V_1, \ldots, V_{i-1}\}$. Then the probability that V_i is a repeat, even conditioned on $V_1, \ldots, V_{i-1}$, is at most $(i-1)/N \leq KD/N$. So,

$$\begin{aligned}\Pr[|N(S)| \leq (D-2) \cdot K] &\leq \Pr\left[\text{there are at least } 2K \text{ repeats among } V_1, \ldots, V_{KD}\right] \\ &\leq \binom{KD}{2K}\left(\frac{KD}{N}\right)^{2K}.\end{aligned}$$

Thus, we find that

$$p_K \leq \binom{N}{K}\binom{KD}{2K}\left(\frac{KD}{N}\right)^{2K} \leq \left(\frac{Ne}{K}\right)^{K}\left(\frac{KDe}{2K}\right)^{2K}\left(\frac{KD}{N}\right)^{2K} = \left(\frac{e^3 D^4 K}{4N}\right)^{K}$$

where e is the base of the natural logarithm. Since $K \leq \alpha N$, we can set $\alpha = 1/(e^3 D^4)$ to obtain $p_K \leq 4^{-K}$. Thus

$$\Pr_{G \in \mathrm{Bip}_{N,D}}[G \text{ is not an } (\alpha N, D-2) \text{ expander}] \leq \sum_{K=1}^{\lfloor \alpha N \rfloor} 4^{-K} < \frac{1}{2}$$

□

There are a number of variants to the above probabilistic construction of expanders.

- We can obtain a bipartite multigraph that is D-regular on both sides by taking the union of D random perfect matchings. This can be analyzed using a small modification of the analysis above; even though $V_1, \ldots, V_{KD}$ are not independent, the probability of a V_i being a repeat conditioned on $V_1, \ldots, V_{i-1}$ can still be bounded by $KD/(N-K)$. Also, the multigraph can be made into a simple graph by eliminating or redistributing edges.
- One can optimize α rather than the expansion factor A, showing that for all constants $\alpha < 1$ and $D > 2$, there exists a constant $A > 1$ such that for all sufficiently large N, a random graph in $\mathrm{Bip}_{N,D}$ is an $(\alpha N, A)$ vertex expander with high probability.
- In fact, a very general tradeoff between D, α, and A is known: a random D-regular N-vertex bipartite multigraph is an $(\alpha N, A)$ vertex expander with high probability if $D > \frac{\mathrm{H}(\alpha)+\mathrm{H}(\alpha A)}{\mathrm{H}(\alpha)-\alpha A \mathrm{H}(1/A)}$, where $\mathrm{H}(p) = p\log(1/p) + (1-p)\log(1/(1-p))$ is the binary entropy function.
- The results can also be extended to unbalanced bipartite graphs (where the right side is smaller than the left), and nonbipartite graphs as well, and both of these cases are important in some applications.

In addition to being natural combinatorial objects, expander graphs have numerous applications in theoretical computer science, including the construction of fault-tolerant networks (indeed, the first papers on expanders were in conferences on telephone networks), sorting in $O(\log n)$ time in parallel, derandomization (as we will see), lower bounds in circuit complexity and proof complexity, error-correcting codes, negative results regarding integrality gaps for linear programming relaxations and metric embeddings, distributed routing, and data structures. For many of these applications, it is not enough to know that a random graph is a good expander — we need *explicit* constructions, i.e. ones that are deterministic and efficient. We view such explicit expanders as "pseudorandom objects" because they are fixed graphs that possess many of the properties of random graphs.

4.1.2 Spectral Expansion

Intuitively, another way of saying that a graph is well-connected is to require that random walks on the graph converge quickly to the stationary distribution. As we have seen in Section 2.4.2, the mixing rate of random walks in turn is captured well by the second largest eigenvalue of the transition matrix, and this turns out to be a very useful measure of expansion for many purposes.

Recall that for an N-vertex regular directed graph G with random-walk matrix M, we define

$$\lambda(G) \stackrel{\text{def}}{=} \max_{\pi} \frac{\|\pi M - u\|}{\|\pi - u\|} = \max_{x \perp u} \frac{\|xM\|}{\|x\|},$$

where $u = (1/N, \ldots, 1/N) \in \mathbb{R}^N$ is the uniform distribution on $[N]$, the first maximum is over all probability distributions $\pi \in [0,1]^N$, and the second maximum is over all vectors $x \in \mathbb{R}^N$ that are orthogonal to u. We write $\gamma(G) \stackrel{\text{def}}{=} 1 - \lambda(G)$ to denote the *spectral gap* of G.

Definition 4.5. For $\gamma \in [0,1]$, we say that a regular digraph G has *spectral expansion* γ if $\gamma(G) \geq \gamma$ (equivalently, $\lambda(G) \leq 1 - \gamma$).[1]

Larger values of $\gamma(G)$ (or smaller values of $\lambda(G)$) correspond to better expansion. Sometimes it is more natural to state results in terms of $\gamma(G)$ and sometimes in terms of $\lambda(G)$. Surprisingly, this linear-algebraic measure of expansion turns out to be equivalent to the combinatorial measure of vertex expansion for common parameters of interest.

One direction is given by the following:

Theorem 4.6 (spectral expansion ⇒ vertex expansion). If G is a regular digraph with spectral expansion $\gamma = 1 - \lambda$ for some $\lambda \in [0,1]$, then, for every $\alpha \in [0,1]$, G is an $\big(\alpha N, 1/((1-\alpha)\lambda^2 + \alpha)\big)$ vertex expander. In particular, G is an $(N/2, 1+\gamma)$ expander.

We prove this theorem using the following two useful statistics of probability distributions.

Definition 4.7. For a probability distribution π, the *collision probability* of π is defined to be the probability that two independent samples from π are equal, namely $\mathrm{CP}(\pi) = \sum_x \pi_x^2$. The *support* of π is $\mathrm{Supp}(\pi) = \{x : \pi_x > 0\}$.

Lemma 4.8. For every probability distribution $\pi \in [0,1]^N$, we have:

(1) $\mathrm{CP}(\pi) = \|\pi\|^2 = \|\pi - u\|^2 + 1/N$, where u is the uniform distribution on $[N]$.
(2) $\mathrm{CP}(\pi) \geq 1/\left|\mathrm{Supp}(\pi)\right|$, with equality iff π is uniform on $\mathrm{Supp}(\pi)$.

Proof. For Part 1, the fact that $\mathrm{CP}(\pi) = \|\pi\|^2$ follows immediately from the definition. Then, writing $\pi = u + (\pi - u)$, and noting that $\pi - u$ is orthogonal to u, we have $\|\pi\|^2 = \|u\|^2 + \|\pi - u\|^2 = 1/N + \|\pi - u\|^2$.

For Part 2, by Cauchy-Schwarz we have

$$1 = \sum_{x \in \mathrm{Supp}(\pi)} \pi_x \leq \sqrt{|\mathrm{Supp}(\pi)|} \cdot \sqrt{\sum_x \pi_x^2} = \sqrt{|\mathrm{Supp}(\pi)|} \cdot \sqrt{\mathrm{CP}(\pi)},$$

with equality iff π is uniform on $\mathrm{Supp}(\pi)$. □

[1] In other sources (including the original lecture notes on which this survey was based), the spectral expansion referred to λ rather than γ. Here we use γ, because it has the more natural feature that larger values of γ correspond to the graph being "more expanding".

Proof of Theorem 4.6. The condition that G has spectral expansion $\gamma = 1 - \lambda$ is equivalent to saying that $\lambda(G) \leq \lambda$. By the definition of $\lambda(G)$ and Part 1 of Lemma 4.8, we have

$$\mathrm{CP}(\pi M) - \frac{1}{N} \leq \lambda^2 \cdot \left(\mathrm{CP}(\pi) - \frac{1}{N}\right)$$

for every probability distribution π. Letting S be any subset of the vertices of size at most αN and π the uniform distribution on S, we have $\mathrm{CP}(\pi) = 1/|S|$ and $\mathrm{CP}(\pi M) \geq 1/\left|\mathrm{Supp}(\pi M)\right| = 1/\left|N(S)\right|$. Thus,

$$\left(\frac{1}{|N(S)|} - \frac{1}{N}\right) \leq \lambda^2 \cdot \left(\frac{1}{|S|} - \frac{1}{N}\right)$$

Solving for $|N(S)|$ and using $N \geq |S|/\alpha$, we obtain $|N(S)| \geq |S|/(\lambda^2(1-\alpha)+\alpha)$, as desired. □

The other direction, i.e. obtaining spectral expansion from vertex expansion, is more difficult (and we will not prove it here).

Theorem 4.9 (vertex expansion ⇒ spectral expansion). For every $\delta > 0$ and $D > 0$, there exists $\gamma > 0$ such that if G is a D-regular $(N/2, 1+\delta)$ vertex expander, then it also has spectral expansion γ. Specifically, we can take $\gamma = \Omega((\delta/D)^2)$.

Note first the dependence on subset size being $N/2$: this is necessary, because a graph can have vertex expansion $(\alpha N, 1 + \Omega(1))$ for $\alpha < 1/2$ and be disconnected (e.g. the disjoint union of two good expanders), thereby having no spectral expansion. Also note that the bound on γ deteriorates as D increases (since $\delta \leq 1$). This is also necessary, because adding edges to a good expander cannot hurt its vertex expansion, but can hurt its spectral expansion.

Still, roughly speaking, these two results show that vertex expansion and spectral expansion are closely related, indeed equivalent for many interesting settings of parameters:

Corollary 4.10. Let $\mathcal{G}$ be an infinite family of D-regular multigraphs, for a constant $D \in \mathbb{N}$. Then the following two conditions are equivalent:

- There is a constant $\delta > 0$ such that every $G \in \mathcal{G}$ is an $(N/2, 1+\delta)$ vertex expander.
- There is a constant $\gamma > 0$ such that every $G \in \mathcal{G}$ has spectral expansion γ.

When people informally use the term "expander," they often mean a family of regular graphs of *constant degree* D satisfying one of the two equivalent conditions above.

However, the two measures are no longer equivalent if one wants to optimize the expansion constants. For vertex expansion, we have already seen that if we allow α to be a small constant (depending on D), then there exist $(\alpha N, A)$ vertex expanders with A very close to $D-1$, e.g. $A = D - 1.01$, and Problem 4.3 shows that A cannot be any larger than $D-1$. The optimal value for the spectral expansion is also well-understood. First note that, by taking $\alpha \to 0$ in Theorem 4.6, a graph with spectral expansion $1-\lambda$ has vertex expansion $A \approx 1/\lambda^2$ for small sets. Thus, a lower bound on λ is $1/\sqrt{D} - o(1)$. In fact, this lower bound can be improved, as shown in the following theorem (and proven in Problem 4.4):

Theorem 4.11. For every constant $D \in \mathbb{N}$, any D-regular, N-vertex multigraph G satisfies $\lambda(G) \geq 2\sqrt{D-1}/D - o(1)$, where the $o(1)$ term vanishes as $N \to \infty$ (and D is held constant).

Surprisingly, there exist explicit constructions giving $\lambda(G) < 2\sqrt{D-1}/D$. Graphs meeting this bound are called *Ramanujan graphs*. Random graphs almost match this bound, as well:

Theorem 4.12. For any constant $D \in \mathbb{N}$, a random D-regular N-vertex graph satisfies $\lambda(G) \leq 2\sqrt{D-1}/D + o(1)$ with probability $1 - o(1)$ where both $o(1)$ terms vanish as $N \to \infty$ (and D is held constant).

Now let us see what these results for spectral expansion imply in the world of vertex expansion. With Ramanujan graphs ($\lambda(G) \leq 2\sqrt{D-1}/D$), the bound from Theorem 4.6 gives a vertex expansion factor of $A \approx D/4$ for small sets. This is not tight, and it is known that Ramanujan graphs actually have vertex expansion $D/2 - o(1)$ for sets of density $o(1)$, which is tight in the sense that there are families of graphs with $\lambda(G) \to 2\sqrt{D-1}/D$ but vertex expansion at most $D/2$. Still, this vertex expansion is not as good as we obtained via the Probabilistic Method (Theorem 4.2), where we achieved vertex expansion $D - O(1)$. This means that we cannot obtain optimal vertex expansion by going through spectral expansion. Similarly, we cannot obtain optimal spectral expansion by going through vertex expansion (because the bound on spectral expansion in Theorem 4.9 necessarily deteriorates as the degree D increases). The conclusion is that vertex and spectral expansion are loosely equivalent, but only if we are not interested in optimizing the constants in the tradeoffs between various parameters (and for some applications these are crucial).

4.1.3 Other Measures of Expansion

In this section, we mention two other useful measures of expansion involving edges crossing cuts in the graph. For two sets $S, T \subset V(G)$, let $e(S,T) = \{(u,v) \in S \times T \mid \{u,v\} \in E\}$. Here (u,v) refers to an *ordered* pair, in contrast to the definition of $\text{cut}(S,T)$ in Section 2.3.4. Thus, we count edges entirely within $S \cap T$ twice, corresponding to both orientations.

Definition 4.13. A D-regular digraph G is a (K, ε) *edge expander* if for all sets S of at most K vertices, the cut $e(S, \overline{S})$ is of size at least $\varepsilon \cdot |S| \cdot D$.

That is, at least an ε fraction of the edges from S lead outside S. (Sometimes edge expansion is defined without the normalization factor of D, only requiring $|e(S, \overline{S})| \geq \varepsilon \cdot |S|$.) When viewed in terms of the random walk on G, the ratio $e(S, \overline{S})/(|S| \cdot D)$ is the probability that, if we condition the stationary distribution on being in S, the random walk leaves S in one step. It turns out that if we fix $K = N/2$, then edge expansion turns out to be even more closely related to spectral expansion than vertex expansion is. Indeed:

Theorem 4.14. (1) If a D-regular, N-vertex digraph G has spectral expansion γ, then G is an $(N/2, \gamma/2)$ edge expander.

(2) If a D-regular, N-vertex digraph G is a $(N/2, \varepsilon)$ edge expander and at least an α fraction of edges leaving each vertex are self-loops for some $\alpha \in [0,1]$, then G has spectral expansion $\alpha \cdot \varepsilon^2/2$.

The condition about self-loops in Part 2 is to ensure that the graph is far from being bipartite (or more generally "periodic" in the sense that all cycle lengths are divisible by some number larger than 1), because a bipartite graph has spectral expansion 0 but can have positive edge expansion. For graphs with a constant fraction of self-loops at each vertex, the theorem implies that the edge expansion is bounded away from 0 iff the spectral expansion is bounded away from 0. Unlike Corollary 4.10, this equivalence holds even for graphs of unbounded degree. The intuition for the relation is that a large edge expansion ε implies that the random walk on the graph has no "bottlenecks" and thus should mix rapidly. This connection also holds for Markov chains in general (when the definitions are appropriately generalized), where the edge expansion is known as the *conductance*. Part 1 of Theorem 4.14 will follow as a special case of the Expander Mixing Lemma below; we omit the proof of Part 2.

Next, we consider a generalization of edge expansion, where we look at edges not just from a set S to its complement but between any two sets S and T. If we think of an expander as being like a random graph, we would expect the fraction of graph edges that are in $e(S,T)$ to be approximately equal to the product of the densities of S and T. The following result shows that this intuition is correct:

Lemma 4.15 (Expander Mixing Lemma). Let G be a D-regular, N-vertex digraph with spectral expansion $1-\lambda$. Then for all sets of vertices S, T of densities $\alpha = |S|/N$ and $\beta = |T|/N$, we have

$$\begin{aligned} \left| \frac{|e(S,T)|}{N \cdot D} - \alpha\beta \right| &\leq \lambda \sqrt{\alpha \cdot (1-\alpha) \cdot \beta \cdot (1-\beta)}. \\ &\leq \lambda \sqrt{\alpha\beta} \leq \lambda. \end{aligned}$$

Observe that the denominator $N \cdot D$ counts all edges of the graph (as ordered pairs). The lemma states that the difference between the fraction of edges in $e(S,T)$ and the expected value if we were to choose G randomly is "small", roughly λ times the square root of this fraction. Finally, note that Part 1 of Theorem 4.14 follows from the Expander Mixing Lemma by setting $T = S^c$, so $\beta = 1-\alpha$ and $|e(S,T)|/ND \geq (1-\lambda) \cdot \alpha \cdot (1-\alpha) \geq \gamma\alpha/2$.

When a digraph $G = (V,E)$ has the property that $||e(S,T)|/|E| - \alpha\beta| = o(1)$ for all sets S, T (with densities α, β), the graph is called *quasirandom*. Thus, the Expander Mixing Lemma implies that a regular digraph with $\lambda(G) = o(1)$ is quasirandom. Quasirandomness has been studied extensively for dense graphs, in which case it has numerous equivalent formulations. Here we are most interested in sparse graphs, especially constant-degree graphs (for which $\lambda(G) = o(1)$ is impossible).

Proof. Let χ_S be the characteristic (row) vector of S and χ_T the characteristic vector of T. Let A be the adjacency matrix of G, and $M = A/D$ be the random-walk matrix for G. Note that $|e(S,T)| = \chi_S A \chi_T^t = \chi_S (DM) \chi_T^t$, where the superscript t denotes the transpose.

We can express χ_S as the sum of two components, one parallel to the uniform distribution u, and the other a vector $\chi_S^\perp$, where $\chi_S^\perp \perp u$. The coefficient of u is $\langle \chi_S, u\rangle / \langle u, u\rangle = \sum_i (\chi_S)_i = |S| = \alpha N$. Then $\chi_S = (\alpha N)u + \chi_S^\perp$ and similarly $\chi_T = (\beta N)u + \chi_T^\perp$. Intuitively, the components parallel to the uniform distribution "spread" the weight of S and T uniformly over the entire graph, and $\chi_S^\perp$ and $\chi_T^\perp$ will yield the error term.

Formally, we have

$$\begin{aligned}\frac{|e(S,T)|}{N\cdot D} &= \frac{1}{N}((\alpha N)u + \chi_S^\perp)M((\beta N)u + \chi_T^\perp)^t \\ &= \frac{1}{N}(\alpha\beta N^2)uMu^t + \frac{1}{N}(\alpha N)uM(\chi_T^\perp)^t + \frac{1}{N}(\beta N)\chi_S^\perp Mu^t + \frac{1}{N}\chi_S^\perp M(\chi_T^\perp)^t.\end{aligned}$$

Since $uM = u$ and $Mu^t = u^t$, and both $\chi_S^\perp$ and $\chi_T^\perp$ are orthogonal to u, the above expression simplifies to:

$$\frac{|e(S,T)|}{N\cdot D} = (\alpha\beta N)uu^t + \frac{\chi_S^\perp M(\chi_T^\perp)^t}{N} = \alpha\beta + \frac{(\chi_S^\perp M)(\chi_T^\perp)^t}{N}.$$

Thus,

$$\begin{aligned}\left|\frac{|e(S,T)|}{N\cdot D} - \alpha\beta\right| &= \left|\frac{(\chi_S^\perp M)(\chi_T^\perp)^t}{N}\right| \\ &\le \frac{1}{N}\cdot\|\chi_S^\perp M\|\cdot\|\chi_T^\perp\| \\ &\le \frac{1}{N}\cdot\lambda\|\chi_S^\perp\|\cdot\|\chi_T^\perp\|.\end{aligned}$$

To complete the proof, we note that

$$\alpha N = \|\chi_S\|^2 = \|(\alpha N)u\|^2 + \|\chi_S^\perp\|^2 = \alpha^2 N + \|\chi_S^\perp\|^2,$$

so $\|\chi_S^\perp\| = \sqrt{(\alpha-\alpha^2)N} = \sqrt{\alpha\cdot(1-\alpha)\cdot N}$ and similarly $\chi_T^\perp = \sqrt{\beta\cdot(1-\beta)\cdot N}$. □

Similarly to vertex expansion and edge expansion, a natural question is to what extent the converse holds. That is, if $|e(S,T)|/ND$ is always "close" to the product of the densities of S and T, then is $\lambda(G)$ necessarily small? This is indeed true:

Theorem 4.16 (converse to Expander Mixing Lemma). Let G be a D-regular, N-vertex undirected graph. Suppose that for all pairs of disjoint vertex sets S, T, we have $\left|\frac{|e(S,T)|}{N\cdot D} - \mu(S)\mu(T)\right| \le \theta\sqrt{\mu(S)\mu(T)}$ for some $\theta \in [0,1]$, where $\mu(R) = |R|/N$ for any set R of vertices. Then $\lambda(G) = O(\theta\log(1/\theta))$.

Putting the two theorems together, we see that λ and θ are the same up to a logarithmic factor. Thus, unlike the other connections we have seen, this connection is good for highly expanding graphs (i.e. $\lambda(G)$ close to zero, $\gamma(G)$ close to 1).

4.2 Random Walks on Expanders

From the previous section, we know that one way of characterizing an expander graph G is by having a bound on its second eigenvalue $\lambda(G)$, and in fact there exist constant-degree expanders

where $\lambda(G)$ is bounded by a constant less than 1. From Section 2.4.3, we know that this implies that the random walk on G converges quickly to the uniform distribution. Specifically, a walk of length t started at any vertex ends at ℓ_2 distance at most λ^t from the uniform distribution. Thus after $t = O(\log N)$ steps, the distribution is very close to uniform, e.g. the probability of every vertex is $(1 \pm .01)/N$. Note that, if G has constant degree, the number of random bits invested here is $O(t) = O(\log N)$, which is within a constant factor of optimal; clearly $\log N - O(1)$ random bits are also necessary to sample an almost uniform vertex. Thus, expander walks give a very good tradeoff between the number of random bits invested and the "randomness" of the final vertex in the walk. Remarkably, expander walks give good randomness properties not only for the final vertex in the walk, but also for the *sequence* of vertices traversed in the walk. Indeed, in several ways to be formalized below, this sequence of vertices "behaves" like uniform independent samples of the vertex set.

A canonical application of expander walks is for randomness-efficient error reduction of randomized algorithms: Suppose we have an algorithm with constant error probability, which uses some m random bits. Our goal is to reduce the error to 2^{-k}, with a minimal penalty in random bits and time. Independent repetitions of the algorithm suffers just an $O(k)$ multiplicative penalty in time, but needs $O(km)$ random bits. We have already seen that with pairwise independence we can use just $O(m+k)$ random bits, but the time blows up by $O(2^k)$. Expander graphs let us have the best of both worlds, using just $m + O(k)$ random bits, and increasing the time by only an $O(k)$ factor. Note that for $k = o(m)$, the number of random bits is $(1+o(1)) \cdot m$, even better than what pairwise independence gives.

The general approach is to consider an expander graph with vertex set $\{0,1\}^m$, where each vertex is associated with a setting of the random bits. We will choose a uniformly random vertex v_1 and then do a random walk of length $t-1$, visiting additional vertices $v_2, \ldots, v_t$. (Note that unlike the rapid mixing analysis, here we start at a uniformly random vertex.) This requires m random bits for the initial choice, and $\log D$ for each of the $t-1$ steps. For every vertex v_i on the random walk, we will run the algorithm with v_i as the setting of the random coins.

First, we consider the special case of randomized algorithms with one-sided error (**RP**). For these, we should accept if at least one execution of the algorithm accepts, and reject otherwise. If the input is a NO instance, the algorithm never accepts, so we also reject. If the input is a YES instance, we want our random walk to hit at least one vertex that makes the algorithm accept. Let B denote the set of "bad" vertices giving coin tosses that make the algorithm reject. By the definition of **RP**, the density of B is at most $1/2$. Thus, our aim is to show that the probability that *all* the vertices in the walk $v_1, \ldots, v_t$ are in B vanishes exponentially fast in t, if G is a good expander.

The case $t = 2$ follows from the Expander Mixing Lemma given in the previous section. If we choose a random edge in a graph with spectral expansion $1-\lambda$, the probability that both endpoints are in a set B is at most $\mu(B)^2 + \lambda \cdot \mu(B)$. So if $\lambda \ll \mu(B)$, then the probability is roughly $\mu(B)^2$, just like two independent random samples. The case of larger t is given by the following theorem.

Theorem 4.17 (Hitting Property of Expander Walks). If G is a regular digraph with spectral expansion $1 - \lambda$, then for any $B \subset V(G)$ of density μ, the probability that a random walk

$(V_1, \ldots, V_t)$ of $t-1$ steps in G starting in a uniformly random vertex V_1 always remains in B is

$$\Pr\left[\bigwedge_{i=1}^{t} V_i \in B\right] \leq (\mu + \lambda \cdot (1-\mu))^t$$

.

Equivalently, a random walk "hits" the complement of B with high probability. Note that if μ and λ are constants less than 1, then the probability of staying in B is $2^{-\Omega(t)}$, completing the analysis of the efficient error-reduction algorithm for **RP**.

Before proving the theorem, we discuss general approaches to analyzing spectral expanders and random walks on them. Typically, the first step is to express the quantities of interest linear-algebraically, involving applications of the random-walk (or adjacency) matrix M to some vectors v. For example, when proving the Expander Mixing Lemma (Lemma 4.15), we expressed the fraction of edges between sets S and T as $\chi_S M \chi_T^t$ (up to some normalization factor). Then we can proceed in one of the two following ways:

Vector Decomposition Decompose the input vector(s) v as $v = v^{\parallel} + v^{\perp}$, where $v^{\parallel} = (\langle v, u\rangle / \langle u, u\rangle)u$ is the component of v in the direction of the uniform distribution u and $v^{\perp}$ is the component of v orthogonal to u. Then this induces a similar orthogonal decomposition of the output vector vM into

$$vM = (vM)^{\parallel} + (vM)^{\perp} = v^{\parallel}M + v^{\perp}M,$$

where $v^{\parallel}M = v^{\parallel}$ and $\|v^{\perp}M\| \leq \lambda \cdot \|v^{\perp}\|$. Thus, from information about how v's lengths are divided into the uniform and non-uniform components, we deduce information about how vM is divided into the uniform and non-uniform components. This is the approach we took in the proof of the Expander Mixing Lemma.

Matrix Decomposition This corresponds to a different decomposition of the output vector vM that can be expressed in a way that is independent of the decomposition of the input vector v. Specifically, if G has spectral expansion $\gamma = 1 - \lambda$, then

$$vM = v^{\parallel} + v^{\perp}M = \gamma \cdot v^{\parallel} + (\lambda \cdot v^{\parallel} + v^{\perp}M) = \gamma \cdot vJ + \lambda \cdot vE = v(\gamma J + \lambda E),$$

where J is the matrix in which every entry is $1/N$ and the error matrix E satisfies $\|vE\| \leq \|v\|$. The advantage of this decomposition is that we can apply it even when we have no information about how v decomposes (only its length). The fact that M is a convex combination of J and E means that we can often treat each of these components separately and then just apply the triangle inequality. However, it is less refined than the vector decomposition approach, and sometimes gives weaker bounds. Indeed, if we used it to prove the Expander Mixing Lemma (without decomposing χ_S and χ_T), we would get a slightly worse error term of $\lambda\sqrt{\mu(S)\mu(T)} + \lambda\mu(S)\mu(T)$.

The Matrix Decomposition Approach can be formalized using the following notion.

Definition 4.18. The *(spectral) norm* of an $N \times N$ real matrix M is defined to be

$$\|M\| = \max_{x \in \mathbb{R}^N} \frac{\|xM\|}{\|x\|}$$

(If M is symmetric, then $\|M\|$ equals the *largest* absolute value of any eigenvalue of M.)

Some basic properties of the matrix norm are that $\|cA\| = |c| \cdot \|A\|$, $\|A+B\| \leq \|A\| + \|B\|$, and $\|A \cdot B\| \leq \|A\| \cdot \|B\|$ for every two matrices A, B, and $c \in \mathbb{R}$. Following the discussion above, we have the following lemma:

Lemma 4.19. Let G be a regular digraph on N vertices with random-walk matrix M. Then G has spectral expansion $\gamma = 1-\lambda$ iff $M = \gamma J + \lambda E$, where J is the $N \times N$ matrix where every entry is $1/N$ (i.e. the random-walk matrix for the complete graph with self-loops) and $\|E\| \leq 1$.

Proof. Suppose that G has spectral expansion γ. Then define $E = (M - \gamma J)/\lambda$. To see that E has norm at most 1, first observe that $uE = (uM - \gamma uJ)/\lambda = (1-\gamma)u/\lambda = u$. Thus it suffices to show that for every vector v orthogonal to u, the vector vE is orthogonal to u and is of length at most $\|v\|$. Orthogonality follows because vM is orthogonal to u (by regularity of G) and $vJ = 0$. The length bounds follows from $vE = (vM)/\lambda$ and $\|vM\| \leq \lambda\|v\|$ by the spectral expansion of G.

Conversely, suppose that $M = \gamma J + \lambda E$ for some $\|E\| \leq 1$. Then for every vector v orthogonal to u, we have $\|vM\| = \|0 + \lambda vE\| \leq \lambda\|v\|$, and thus G has spectral expansion γ. □

Intuitively, this lemma says that we can think of a random step on a graph with spectral expansion γ as being a random step on the complete graph with probability γ and "not doing damage" with probability $1-\gamma$. This intuition would be completely accurate if E were a stochastic matrix, but it is typically not (e.g. it may have negative entries). Still, note that the bound given in Theorem 4.17 exactly matches this intuition: in every step, the probability of remaining in B is at most $\gamma \cdot \mu + \lambda = \mu + \lambda \cdot (1-\mu)$.

Now we can return to the proof of the theorem.

Proof Theorem 4.17. We need a way to express getting stuck in B linear-algebraically. For that, we define P to be the diagonal matrix with $P_{i,i} = 1$ if $i \in B$ and $P_{i,i} = 0$ otherwise. Thus, the probability a distribution π picks a node in B is $|\pi P|_1$, where $|\cdot|_1$ is the ℓ_1 norm, $|x|_1 = \sum |x_i|$ (which in our case is equal to the sum of the components of the vector, since all values are nonnegative).

Let M be the random-walk matrix of G. The probability distribution for the first vertex V_1 is given by the vector u. Now we can state the following crucial fact:

Claim 4.20. The probability that the random walk stays entirely within B is precisely $|uP(MP)^{t-1}|_1$.

> **Proof of claim:** By induction on ℓ, we show that $\left(uP(MP)^{\ell}\right)_i$ is the probability that the first $\ell+1$ vertices of the random walk are in B and the $(\ell+1)$'st vertex is i. The base case is $\ell = 0$. If $i \in B$, $(uP)_i = 1/N$; if $i \notin B$, $(uP)_i = 0$. Now assume the hypothesis holds up to some ℓ. Then $\left(uP(MP)^{\ell}M\right)_i$ is the probability that the first $\ell+1$ vertices of the random walk are in B and the $(\ell+2)$'nd vertex is i (which may or may not be in B). Multiplying by P, we zero out all components for nodes not in B and leave the others unchanged. Thus, we obtain the probability that the first $\ell+2$ vertices are in B and the $(\ell+2)$'nd vertex is i. □

To get a bound in terms of the spectral expansion, we will now switch to the ℓ_2 norm. The intuition is that multiplying by M shrinks the component that is perpendicular to u (by expansion) and multiplying by P shrinks the component parallel to u (because it zeroes out some entries). Thus, we should be able to show that the norm $\|MP\|$ is strictly less than 1. Actually, to get the best bound, we note that $uP(MP)^{t-1} = uP(PMP)^{t-1}$, because $P^2 = P$, so we instead bound $\|PMP\|$. Specifically:

Claim 4.21. $\|PMP\| \leq \mu + \lambda \cdot (1 - \mu)$.

Proof of claim: Using the Matrix Decomposition Lemma (Lemma 4.19), we have:

$$\begin{aligned} \|PMP\| &= \|P(\gamma J + \lambda E)P\| \\ &\leq \gamma \cdot \|PJP\| + \lambda \cdot \|PEP\| \\ &\leq \gamma \cdot \|PJP\| + \lambda \end{aligned}$$

Thus, we only need to analyze the case of J, the random walk on the complete graph. Given any vector x, let $y = xP$, so

$$xPJP = yJP = \left(\sum_i y_i\right) uP.$$

Since $\|y\| \leq \|x\|$ and y has at most μN nonzero coordinates, we have

$$\|xPJP\| \leq \left|\sum_i y_i\right| \cdot \|uP\| \leq \left(\sqrt{\mu N} \cdot \|y\|\right) \cdot \sqrt{\frac{\mu}{N}} \leq \mu \cdot \|x\|.$$

Thus,

$$\|PMP\| \leq \gamma \cdot \mu + \lambda = \mu + \lambda \cdot (1 - \mu).$$

□

Using Claims 4.20 and 4.21, the probability of never leaving B in a $(t-1)$-step random walk is

$$\begin{aligned} |uP(MP)^{t-1}|_1 &\leq \sqrt{\mu N} \cdot \|uP(MP)^{t-1}\| \\ &\leq \sqrt{\mu N} \cdot \|uP\| \cdot \|PMP\|^{t-1} \\ &\leq \sqrt{\mu N} \cdot \sqrt{\frac{\mu}{N}} \cdot (\mu + \lambda \cdot (1 - \mu))^{t-1} \\ &\leq (\mu + \lambda \cdot (1 - \mu))^t. \end{aligned}$$

□

The hitting properties described above suffice for reducing the error of **RP** algorithms. What about **BPP** algorithms, which have two-sided error? They are handled by the following.

Theorem 4.22 (Chernoff Bound for Expander Walks). Let G be a regular digraph with N vertices and spectral expansion $1-\lambda$, and let $f : [N] \to [0,1]$ be any function. Consider a random walk $V_1, \ldots, V_t$ in G from a uniform start vertex V_1. Then for any $\varepsilon > 0$

$$\Pr\left[\left|\frac{1}{t}\sum_i f(V_i) - \mu(f)\right| \geq \lambda + \varepsilon\right] \leq 2e^{-\Omega(\varepsilon^2 t)}.$$

Note that this is just like the standard Chernoff Bound (Theorem 2.21), except that our additive approximation error increases by $\lambda = 1-\gamma$. Thus, unlike the Hitting Property we proved above, this bound is only useful when λ is sufficiently small (as opposed to bounded away from 1). This can be achieved by taking an appropriate power of the initial expander. However, there is a better Chernoff Bound for Expander Walks, where λ does not appear in the approximation error, but the exponent in the probability of error is $\Omega(\gamma\varepsilon^2 t)$ instead of $\Omega(\varepsilon^2 t)$. The bound above suffices in the common case that a small constant approximation error can be tolerated, as in error reduction for **BPP**.

Proof. Let X_i be the random variable $f(V_i)$, and $X = \sum_i X_i$. Just like in the standard proof of the Chernoff Bound (Problem 2.7), we show that the expectation of the moment generating function $e^{rX} = \prod_i e^{rX_i}$ is not much larger than $e^{r\,\mathrm{E}[X]}$ and apply Markov's Inequality, for a suitable choice of r. However, here the factors e^{rX_i} are not independent, so the expectation does not commute with the product. Instead, we express $\mathrm{E}[e^{rX}]$ linear-algebraically as follows. Define a diagonal matrix P whose (i,i)'th entry is $e^{rf(i)}$. Then, similarly Claim 4.20 in the proof of the hitting proof above, we observe that

$$\mathrm{E}[e^{rX}] = \left|uP(MP)^{t-1}\right|_1 = \left|u(MP)^t\right|_1 \leq \sqrt{N}\cdot\|u\|\cdot\|MP\|^t = \|MP\|^t.$$

To see this, we simply note that each cross-term in the matrix product $uP(MP)^{t-1}$ corresponds to exactly one expander walk $v_1, \ldots, v_t$, with a coefficient equal to the probability of this walk times $\prod_i e^{f(v_i)}$. By the Matrix Decomposition Lemma (Lemma 4.19), we can bound

$$\|MP\| \leq (1-\lambda)\cdot\|JP\| + \lambda\cdot\|EP\|.$$

Since J simply projects onto the uniform direction, we have

$$\begin{aligned}
\|JP\|^2 &= \frac{\|uP\|^2}{\|u\|^2} \\
&= \frac{\sum_v (e^{r\cdot f(v)}/N)^2}{\sum_v (1/N)^2} \\
&= \frac{1}{N}\cdot\sum_v e^{2rf(v)} \\
&= \frac{1}{N}\cdot\sum_v (1 + 2rf(v) + O(r^2)) \\
&= 1 + 2r\mu + O(r^2)
\end{aligned}$$

for $r \leq 1$, and thus

$$\|JP\| = \sqrt{1 + 2r\mu + O(r^2)} = 1 + r\mu + O(r^2).$$

For the error term, we have

$$\|EP\| \leq \|P\| \leq e^r = 1 + r + O(r^2).$$

Thus,

$$\|MP\| \leq (1-\lambda) \cdot (1 + r\mu + O(r^2)) + \lambda \cdot (1 + r + O(r^2)) \leq 1 + (\mu + \lambda)r + O(r^2),$$

and we have

$$\mathrm{E}[e^{rX}] \leq (1 + (\mu+\lambda)r + O(r^2))^t \leq e^{(\mu+\lambda)rt + O(r^2 t)}.$$

By Markov's Inequality,

$$\Pr[X \geq (\mu + \lambda + \varepsilon) \cdot t] \leq e^{-\varepsilon rt + O(r^2 t)} = e^{-\Omega(\varepsilon^2 t)},$$

if we set $r = \varepsilon/c$ for a large enough constant c. By applying the same analysis to the function $1 - f$, we see that $\Pr[X \leq (\mu - \lambda - \varepsilon)t] = e^{-\Omega(\varepsilon^2 t)}$, and this establishes the theorem. □

We now summarize the properties that expander walks give us for randomness-efficient error reduction and sampling.

For reducing the error of a **BPP** algorithm from 1/3 to 2^{-k}, we can apply Theorem 4.22 with $\lambda = \varepsilon = 1/12$, so that a walk of length $t = O(k)$ suffices. If the original **BPP** algorithm used m random bits and the expander is of constant degree (which is possible with $\lambda = 1/12$), then the number of random bits needed is only $m + O(k)$. Comparing with previous methods for error reduction, we have:

	Number of Repetitions	Number of Random Bits
Independent Repetitions	$O(k)$	$O(km)$
Pairwise Independent Repetitions	$O(2^k)$	$O(k+m)$
Expander Walks	$O(k)$	$m + O(k)$

For Sampling, where we are given an oracle to a function $f : \{0,1\}^m \to [0,1]$ and we want to approximate $\mu(f)$ to within an additive error of ε, we can apply Theorem 4.22 with error $\varepsilon/2$ and $\lambda = \varepsilon/2$. The needed expander can be obtained by taking an $O(\log(1/\varepsilon))$'th power of a constant-degree expander, yielding the following bounds:

	Number of Samples	Number of Random Bits
Truly Random Sample	$O((1/\varepsilon^2) \cdot \log(1/\delta))$	$O(m \cdot (1/\varepsilon^2) \cdot \log(1/\delta)$
Pairwise Independent Samples	$O((1/\varepsilon^2) \cdot (1/\delta))$	$O(m + \log(1/\varepsilon) + \log(1/\delta)$
Expander Walks	$O((1/\varepsilon^2) \cdot \log(1/\delta))$	$m + O(\log(1/\delta) \cdot (\log(1/\varepsilon)/\varepsilon^2))$

The $\log(1/\varepsilon)$ factor in the number of random bits comes because we took an $O(\log(1/\varepsilon))$'th power of a constant-degree expander and thus spend $O(\log(1/\varepsilon))$ random bits for each step on the expander. This is actually not necessary and comes from the slightly weaker Chernoff Bound we proved. In any case, note that expander walks have a much better dependence on the error probability δ in the number of samples (as compared to pairwise independence), but have a worse dependence on the approximation error ε in the number of random bits. Problem 4.5 shows how to combine these two samplers to achieve the best of both.

Similarly to pairwise independence, the sampling algorithm based on expander walks is actually an averaging sampler in the sense of Definition 3.29:

Theorem 4.23 (Expander-Walk Sampler). For every $m \in \mathbb{N}$ and $\delta, \varepsilon \in [0,1]$, there is a (δ, ε) averaging sampler $\mathrm{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ using $n = m + O(\log(1/\delta) \cdot \log(1/\varepsilon)/\varepsilon^2)$ random bits and $t = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples.

The sampling algorithm of Problem 4.5 that combines expander walks and pairwise independence, however, is *not* an averaging sampler, and it is an open problem to achieve similar parameters with an explicit averaging sampler:

Open Problem 4.24. Give an explicit construction of a (δ, ε) averaging sampler $\mathrm{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ that uses $n = O(m + \log(\delta) + \log(1/\varepsilon))$ random bits and $t = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples.

Before we end this section, we make an important remark: we have not actually given an efficient algorithm for randomness-efficient error reduction (or an explicit averaging sampler)! Our algorithm assumes an expander graph of exponential size, namely 2^m where m is the number of random bits used by the algorithm. Generating such a graph at random would use far too many coins. Even generating it deterministically would not suffice, since we would have to write down an exponential-size object. In the following section, we will see how to define an explicit expander graph without writing it down in its entirety, and efficiently do random walks in such a graph.

4.3 Explicit Constructions

As discussed in previous sections, expander graphs have numerous applications in theoretical computer science. (See also the Chapter Notes and Exercises.) For some of these applications, it may be acceptable to simply choose the graph at random, as we know that a random graph will be a good expander with high probability. For many applications, however, this simple approach does not suffice. Some reasons are the following (in increasing order of significance):

- We may not want to tolerate the error probability introduced by the (unlikely) event that the graph is not an expander. To deal with this, we could try checking that the graph is an expander. Computing expansion is **NP**-hard for most of the combinatorial measures (e.g. vertex expansion or edge expansion) of a given graph is **NP**-hard, but the spectral expansion can be computed to high precision in time polynomial in the size of the graph (as it is just an eigenvalue computation). As we saw, spectral expansion does yield estimates on vertex expansion and edge expansion (but cannot give optimal expansion in these measures).
- Some of the applications of expanders (like the one from the previous section) are for reducing the amount of randomness needed for certain tasks. Thus choosing the graph at random defeats the purpose.
- A number of the applications require *exponentially large* expander graphs, and thus we cannot even write down a randomly chosen expander. For example, for randomness-efficient error reduction of randomized algorithms, we need an expander on 2^m nodes where m is the number of random bits used by the algorithm.

From a more philosophical perspective, finding explicit constructions is a way of developing and measuring our understanding of these important combinatorial objects.

A couple of alternatives for defining explicit constructions of expanders on N nodes are:

Mildly Explicit: Construct a complete representation of the graph in time poly(N).
Fully Explicit: Given a node $u \in [N]$ and $i \in [D]$, where D is the degree of the expander, compute the i^{th} neighbor of u in time poly($\log N$).

Consider the randomness-efficient error reduction application discussed in the previous section, in which we performed a random walk on an expander graph with exponentially many nodes. Mild explicitness is insufficient for this application, as the desired expander graph is of exponential size, and hence cannot be even entirely stored, let alone constructed. But full explicitness is perfectly suited for efficiently conducting a random walk on a huge graph. So now our goal is the following:

Goal: Devise a fully explicit construction of an infinite family $\{G_i\}$ of D-regular graphs with spectral expansion at least γ, where D and $\gamma > 0$ are constants independent of i.

We remark that we would also like the set $\{N_i\}$, where N_i is the number of vertices in G_i, to be not too sparse, so that the family of graphs $\{G_i\}$ has graphs of size close to any desired size.

4.3.1 Algebraic Constructions

Here we mention a few known explicit constructions that are of interest because of their simple description, the parameters achieved, and/or the mathematics that goes into their analysis. We will not prove the expansion properties of any of these constructions (but will rather give a different explicit construction in the subsequent sections).

Construction 4.25 (discrete torus expanders). Let $G = (V, E)$ be the graph with vertex set $V = \mathbb{Z}_M \times \mathbb{Z}_M$, and edges from each node (x, y) to the nodes (x, y), $(x+1, y)$, $(x, y+1)$, $(x, x+y)$, $(-y, x)$, where all arithmetic is modulo M.

This is a fully explicit 5-regular digraph with $N = M^2$ nodes and spectral expansion $\gamma = \Omega(1)$. It can be made undirected by adding a reverse copy of each edge. We refer to these as "discrete torus" expanders because $\mathbb{Z}_M^2$ can be viewed as a discrete version of the real torus, namely $[0, 1]^2$ with arithmetic modulo 1. The expansion of these graphs was originally proved using group representation theory, but later proofs for similar discrete-torus expanders were found that only rely on Fourier analysis over the torus.

Construction 4.26 (p-cycle with inverse chords). This is the graph $G = (V, E)$ with vertex set $V = \mathbb{Z}_p$ and edges that connect each node x with the nodes: $x + 1$, $x - 1$ and x^{-1} (where all arithmetic is mod p and we define 0^{-1} to be 0).

This graph is only mildly explicit since we do not know how to construct n-bit primes deterministically in time poly(n) (though it is conjectured that we can do so by simply checking the first poly(n) n-bit numbers). The proof of expansion relies on the "Selberg 3/16 Theorem" from number theory.

Construction 4.27 (Ramanujan graphs). $G = (V, E)$ is a graph with vertex set $V = \mathbb{F}_q \cup \{\infty\}$, the finite field of prime order q s.t. $q \equiv 1 \bmod 4$ plus one extra node representing infinity. The edges in this graph connect each node z with all z' of the form

$$z' = \frac{(a_0 + ia_1)z + (a_2 + ia_3)}{(-a_2 + ia_3)z + (a_0 - ia_1)}$$

for $a_0, a_1, a_2, a_3 \in \mathbb{Z}$ such that $a_0^2 + a_1^2 + a_2^2 + a_3^2 = p$, a_0 is odd and positive, and a_1, a_2, a_3 are even, for some fixed prime $p \neq q$ such that $p \equiv 1 \bmod 4$, q is a square modulo p, and $i \in \mathbb{F}_q$ such that $i^2 = -1 \bmod q$.

The degree of the graph is the number of solutions to the equation $a_0^2 + a_1^2 + a_2^2 + a_3^2 = p$, which turns out to be $D = p + 1$, and it has $\lambda(G) \leq 2\sqrt{D-1}/D$, so it is an *optimal* spectral expander. (See Theorems 4.11 and 4.12, and note that this bound is even better than we know for random graphs, which have an additive $o(1)$ term in the spectral expansion.) These graphs are also only mildly explicit, again due to the need to find the prime q.

These are called Ramanujan Graphs because the proof of their spectral expansion relies on results in number theory concerning the "Ramanujan Conjectures." Subsequently, the term *Ramanujan graphs* came to refer to any infinite family of graphs with optimal spectral expansion $\gamma \geq 1 - 2\sqrt{D-1}/D$.

4.3.2 Graph Operations

The explicit construction of expanders given in the next section will be an iterative one, where we start with a "constant size" expander H and repeatedly apply graph operations to get bigger expanders. The operations that we apply should increase the number of nodes in the graph, while keeping the degree and the second eigenvalue λ bounded. We'll see three operations, each improving one property while paying a price on the others; however, combined together, they yield the desired expander. It turns out that this approach for constructing expanders will also be useful in derandomizing the logspace algorithm for Undirected S-T Connectivity, as we will see in Section 4.4.

The following concise notation will be useful to keep track of each of the parameters:

Definition 4.28. An (N, D, γ)-*graph* is a D-regular digraph on N vertices with spectral expansion γ.

4.3.2.1 Squaring

~~**Definition 4.29 (Squaring of Graphs).** If $G = (V, E)$ is a D-regular digraph, then $G^2 = (V, E')$~~ is a D^2-regular digraph on the same vertex set, where the (i, j)'th neighbor of a vertex x is the j'th neighbor of the i'th neighbor of x. In particular, a random step on G^2 consists of two random steps on G.

Lemma 4.30. If G is a $(N, D, 1-\lambda)$-graph, then G^2 is a $(N, D^2, 1-\lambda^2)$-graph.

Namely, the degree deteriorates by squaring, while the spectral expansion is improved from $\gamma = 1-\lambda$ to $\gamma' = 1-\lambda^2 = 2\gamma - \gamma^2$.

Proof. The effect of squaring on the number of nodes N and the degree D is immediate from the definition. For the spectral expansion, note that if M is the random-walk matrix for G, then M^2 is the random-walk matrix for G^2. So for any vector $x \perp u$,

$$\|xM^2\| \leq \lambda \cdot \|xM\| \leq \lambda^2 \cdot \|x\|.$$

□

4.3.2.2 Tensoring

The next operation we consider increases the size of the graph at the price of increasing the degree.

Definition 4.31 (Tensor Product of Graphs). Let $G_1 = (V_1, E_1)$ be D_1-regular and $G_2 = (V_2, E_2)$ be D_2-regular. Then their *tensor product* is the D_1D_2-regular graph $G_1 \otimes G_2 = (V_1 \times V_2, E)$, where the (i_1, i_2)'th neighbor of a vertex (x_1, x_2) is (y_1, y_2), where y_b is the i_b'th neighbor of x_b in G_b. That is, a random step on $G_1 \otimes G_2$ consists of a random step on G_1 in the first component and a random step on G_2 in the second component.

Often this operation is simply called the "product" of G_1 and G_2, but we use "tensor product" to avoid confusion with squaring and to reflect its connection with the standard tensor products in linear algebra:

Definition 4.32 (Tensor Products of Vectors and Matrices). Let $x \in \mathbb{R}^{N_1}, y \in \mathbb{R}^{N_2}$, then their *tensor product* is the vector $z = x \otimes y \in \mathbb{R}^{N_1N_2}$ where $z_{ij} = x_i y_j$.

Similarly, for matrices $A = (a_{ij}) \in \mathbb{R}_{N_1 \times N_1}, B = (b_{ij}) \in \mathbb{R}_{N_2 \times N_2}$, their *tensor product* is the matrix $C = A \otimes B \in \mathbb{R}_{N_1N_2 \times N_1N_2}$ where $C = (c_{ij,i'j'})$ for $c_{ij,i'j'} = a_{ii'}b_{jj'}$.

A few comments on the tensor operation:

- A random walk on a tensor graph $G_1 \otimes G_2$ is equivalent to taking two independent random walks on G_1 and G_2.
- For vectors $x \in \mathbb{R}^{N_1}, y \in \mathbb{R}^{N_2}$ that are probability distributions (i.e. nonnegative vectors with ℓ_1 norm 1), their tensor product $x \otimes y$ is a probability distribution on $[N_1] \times [N_2]$ where the two components are independently distributed according to x and y, respectively.
- $(x \otimes y)(A \otimes B) = (xA) \otimes (yB)$ for every $x \in \mathbb{R}^{N_1}, y \in \mathbb{R}^{N_2}$, and in fact $A \otimes B$ is the unique matrix with this property.
- Not all vectors $z \in \mathbb{R}^{N_1N_2}$ are decomposable as $x \otimes y$ for $x \in \mathbb{R}^{N_1}$ and $y \in \mathbb{R}^{N_2}$. Nevertheless, the set of all decomposable tensors $x \otimes y$ spans $\mathbb{R}^{N_1N_2}$.

- If M_1, M_2 are the random-walk matrices for graphs G_1, G_2 respectively, then the random-walk matrix for the graph $G_1 \otimes G_2$ is

$$M_1 \otimes M_2 = (I_{N_1} \otimes M_2)(M_1 \otimes I_{N_2}) = (M_1 \otimes I_{N_2})(I_{N_1} \otimes M_2),$$

  where I_N denotes the $N \times N$ identity matrix. That is, we can view a random step on $G_1 \otimes G_2$ as being a random step on the G_1 component followed by one on the G_2 component or vice-versa.

The effect of tensoring on expanders is given by the following:

Lemma 4.33. If G_1 is an (N_1, D_1, γ_1)-graph and G_2 is an (N_2, D_2, γ_2)-graph, then $G_1 \otimes G_2$ is an $(N_1N_2, D_1D_2, \min\{\gamma_1, \gamma_2\})$-graph.

In particular, if $G_1 = G_2$, then the number of nodes improves, the degree deteriorates, and the spectral expansion remains unchanged.

Proof. As usual, we write $\gamma_1 = 1 - \lambda_1$, $\gamma_2 = 1 - \lambda_2$; then our goal is to show that $G_1 \otimes G_2$ has spectral expansion $1 - \max\{\lambda_1, \lambda_2\}$. The intuition is as follows. Any probability distribution (V_1, V_2) on the vertices (v_1, v_2) of $G_1 \otimes G_2$ can be thought of as picking a cloud v_1 according to the marginal distribution[2] V_1 and then picking the vertex v_2 within the cloud v_1 according to the conditional distribution $V_2|_{V_1=v_1}$. If the overall distribution on pairs is far from uniform, then either

(1) The marginal distribution V_1 on the clouds must be far from uniform, *or*
(2) the conditional distributions $V_2|_{V_1=v_1}$ within the clouds must be far from uniform.

When we take a random step, the expansion of G_1 will bring us closer to uniform in Case 1 and the expansion of G_2 will bring us closer to uniform in Case 2.

One way to prove the bound in the case of undirected graphs is to use the fact that the eigenvalues of $M_1 \otimes M_2$ are all the products of eigenvalues of M_1 and M_2, so the three largest absolute values are $\{1 \cdot 1, \lambda_1 \cdot 1, 1 \cdot \lambda_2\}$. Instead, we use the Vector Decomposition Method to give a proof that matches the intuition more closely and is a good warm-up for the analysis of the zig-zag product in the next section. Given any vector $x \in \mathbb{R}^{N_1N_2}$ that is orthogonal to $u_{N_1N_2}$, we can decompose x as $x = x^{\|} + x^{\perp}$ where $x^{\|}$ is a multiple of u_{N_2} on each cloud of size N_2 and $x^{\perp}$ is orthogonal to u_{N_2} on each cloud. Note that $x^{\|} = y \otimes u_{N_2}$, where $y \in \mathbb{R}^{N_1}$ is orthogonal to u_{N_1} (because $x^{\|} = x - x^{\perp}$ is orthogonal to $u_{N_1N_2}$). If we think of x as the nonuniform component of a probability distribution, then $x^{\|}$ and $x^{\perp}$ correspond to the two cases in the intuition above.

For the first case, we have

$$x^{\|}M = (y \otimes u_{N_2})(M_1 \otimes M_2) = (yM_1) \otimes u_{N_2}.$$

The expansion of G_1 tells us that M_1 shrinks y by a factor of λ_1, and thus $\|x^{\|}M\| \leq \lambda_1 \cdot \|x^{\|}\|$. For the second case, we write

$$x^{\perp}M = x^{\perp}(I_{N_1} \otimes M_2)(M_1 \otimes I_{N_2}).$$

[2] For two jointly distributed random variables (X, Y), the *marginal distribution* of X is simply the distribution of X alone, ignoring information about Y.

The expansion of G_2 tells us that M_2 will shrink $x^\perp$ by a factor of λ_2 on each cloud, and thus $I_{N_1} \otimes M_2$ will shrink $x^\perp$ by the same factor. The subsequent application of $M_1 \otimes I_{N_2}$ cannot increase the length (being the random-walk matrix for a regular graph, albeit a disconnected one). Thus, $\|x^\perp M\| \leq \lambda_2 \|x^\perp\|$.

Finally, we argue that $x^\parallel M$ and $x^\perp M$ are orthogonal. Note that $x^\parallel M = (yM_1) \otimes u_{N_2}$ is a multiple of u_{N_2} on every cloud. Thus it suffices to argue that $x^\perp$ remains orthogonal to u_{N_2} on every cloud after we apply M. Applying $(I_{N_1} \otimes M_2)$ retains this property (because applying M_2 preserves orthogonality to u_{N_2}, by regularity of G_2) and applying $(M_1 \otimes I_{N_2})$ retains this property because it assigns each cloud a linear combination of several other clouds (and a linear combination of vectors orthogonal to u_{N_2} is also orthogonal to u_{N_2}).

Thus,

$$\begin{aligned} \|xM\|^2 &= \|x^\parallel M\|^2 + \|x^\perp M\|^2 \\ &\leq \lambda_1^2 \cdot \|x^\parallel\|^2 + \lambda_2^2 \cdot \|x^\perp\|^2 \\ &\leq \max\{\lambda_1, \lambda_2\}^2 \cdot (\|x^\parallel\|^2 + \|x^\perp\|^2) \\ &= \max\{\lambda_1, \lambda_2\}^2 \cdot \|x\|^2, \end{aligned}$$

as desired. □

4.3.2.3 The Zig-Zag Product

Of the two operations we have seen, one (squaring) improves expansion and one (tensoring) increases size, but both have the deleterious effect of increasing the degree. Now we will see a third operation that decreases the degree, without losing too much in the expansion. By repeatedly applying these three operations, we will be able to construct arbitrarily large expanders while keeping both the degree and expansion constant.

Let G be an (N_1, D_1, γ_1) expander and H be a (D_1, D_2, γ_2) expander. The *zig-zag product* of G and H, denoted $G \textcircled{z} H$, will be defined as follows. The nodes of $G \textcircled{z} H$ are the pairs (u, i) where $u \in V(G)$ and $i \in V(H)$. The edges of $G \textcircled{z} H$ will be defined so that a random step on $G \textcircled{z} H$ corresponds to a step on G, but using a random step on H to choose the edge in G. (This is the reason why we require the number of vertices in H to be equal to the degree of G.) A step in $G \textcircled{z} H$ will therefore involve a step to a random neighbor in H and then a step in G to a neighbor whose index is equal to the label of the current node in H. Intuitively, a random walk on a "good" expander graph H should generate choices that are sufficiently random to produce a "good' random walk on G. One problem with this definition is that it is not symmetric. That is, the fact that you can go from (u, i) to (v, j) does not mean that you can go from (v, j) to (u, i). We correct this by adding another step in H after the step in G. In addition to allowing us to construct undirected expander graphs, this extra step will also turn out to be important for the expansion of $G \textcircled{z} H$.

More formally,

Definition 4.34 (Zig-zag Product). Let G be an D_1-regular digraph on N_1 vertices, and H a D_2-regular digraph on D_1 vertices. Then $G \textcircled{z} H$ is a graph whose vertices are pairs $(u, i) \in [N_1] \times [D_1]$. For $a, b \in [D_2]$, the (a, b)'th neighbor of a vertex (u, i) is the vertex (v, j) computed as follows:

(1) Let i' be the a'th neighbor of i in H.

(2) Let v be the i''th neighbor of u in G, so $e = (u, v)$ is the i''th edge leaving u. Let j' be such that e is the j''th edge entering v in G. (In an undirected graph, this simply means that u is the j''th neighbor of v.)
(3) Let j be the b'th neighbor of j' in H.

Note that the graph $G \circledz H$ depends on how the edges leaving and entering each vertex of G are numbered. Thus it is best thought of as an operation on labelled graphs. (This is made more explicit in Section 4.3.3 via the notion of an "edge-rotation map.") Nevertheless, the bound we will prove on expansion holds regardless of the labelling:

Theorem 4.35. If G is a (N_1, D_1, γ_1)-graph, and H is a (D_1, D_2, γ_2)-graph then $G \circledz H$ is a $(N_1 D_1, D_2^2, \gamma = \gamma_1 \cdot \gamma_2^2)$-graph. In particular, if $\gamma_1 = 1 - \lambda_1$ and $\gamma_2 = 1 - \lambda_2$, then $\gamma = 1 - \lambda$ for $\lambda \leq \lambda_1 + 2\lambda_2$.

G should be thought of as a big graph and H as a small graph, where D_1 is a large constant and D_2 is a small constant. Note that the number of nodes D_1 in H is required to equal the degree of G. Observe that when $D_1 > D_2^2$ the degree is reduced by the zig-zag product.

There are two different intuitions underlying the expansion of the zig-zag product:

- Given an initial distribution (U, I) on the vertices of $G_1 \circledz G_2$ that is far from uniform, there are two extreme cases, just as in the intuition for the tensor product.[3] Either
  (1) All the (conditional) distributions $I|_{U=u}$ within the clouds are far from uniform, *or*
  (2) All the (conditional) distributions $I|_{U=u}$ within the clouds of size D_1 are uniform (in which case the marginal distribution U on the clouds must be far from uniform).

  In Case 1, the first H-step $(U, I) \mapsto (U, I')$ already brings us closer to the uniform distribution, and the other two steps cannot hurt (as they are steps on regular graphs). In Case 2, the first H-step has no effect, but the G-step $(U, I') \mapsto (V, J')$ has the effect of making the marginal distribution on clouds closer to uniform, i.e. V is closer to uniform than U. But note that the joint distribution (V, J') isn't actually any closer to the uniform distribution on the vertices of $G_1 \circledz G_2$ because the G-step is a permutation. Still, if the marginal distribution V on clouds is closer to uniform, then the conditional distributions within the clouds $J'|_{V=v}$ must have become further from uniform, and thus the second H-step $(V, J') \mapsto (V, J)$ brings us closer to uniform. This leads to a proof by *Vector Decomposition*, where we decompose any vector x that is orthogonal to uniform into components $x^{\|}$ and $x^{\perp}$, where $x^{\|}$ is uniform on each cloud, and $x^{\perp}$ is orthogonal to uniform on each cloud. This approach gives the best known bounds on the spectral expansion of the zig-zag product, but it can be a bit messy since the two components generally do not remain orthogonal after the steps of the zig-zag product (unlike the case of the tensor product, where we were able to show that $x^{\|}M$ is orthogonal to $x^{\perp}M$).

[3] Here we follow our convention of using capital letters to denote random variables corresponding to the lower-case values in Definition 4.34.

- The second intuition is to think of the expander H as behaving "similarly" to the complete graph on D_1 vertices (with self-loops). In the case that H equals the complete graph, then it is easy to see that $G ⓩ H = G \otimes H$. Thus it is natural to apply *Matrix Decomposition*, writing the random-walk matrix for an arbitrary expander H as a convex combination of the random-walk matrix for the complete graph and an error matrix. This gives a very clean analysis, but slightly worse bounds than the Vector Decomposition Method.

We now proceed with the formal proof, following the Matrix Decomposition approach.

Proof of Theorem 4.35. Let A, B, and M be the random-walk matrices for G_1, G_2, and $G_1 ⓩ G_2$, respectively. We decompose M into the product of three matrices, corresponding to the three steps in the definition of $G_1 ⓩ G_2$'s edges. Let $\tilde{B}$ be the transition matrix for taking a random G_2-step on the second component of $[N_1] \times [D_1]$, i.e. $\tilde{B} = I_{N_1} \otimes B$, where I_{N_1} is the $N_1 \times N_1$ identity matrix. Let $\hat{A}$ be the permutation matrix corresponding to the G_1-step. That is, $\hat{A}_{(u,i),(v,j)}$ is 1 iff (u, v) is the i'th edge leaving u and the j'th edge entering v. By the definition of $G_1 ⓩ G_2$, we have $M = \tilde{B}\hat{A}\tilde{B}$.

By the Matrix Decomposition Lemma (Lemma 4.19), $B = \gamma_2 J + (1 - \gamma_2)E$, where every entry of J equals $1/D_1$ and E has norm at most 1. Then $\tilde{B} = \gamma_2\tilde{J} + (1 - \gamma_2)\tilde{E}$, where $\tilde{J} = I_{N_1} \otimes J$ and $\tilde{E} = I_{N_1} \otimes E$ has norm at most 1.

This gives

$$M = \left(\gamma_2\tilde{J} + (1 - \gamma_2)\tilde{E}\right) \hat{A} \left(\gamma_2\tilde{J} + (1 - \gamma_2)\tilde{E}\right) = \gamma_2^2\tilde{J}\hat{A}\tilde{J} + (1 - \gamma_2^2)F,$$

where F has norm at most 1. Now, the key observation is that $\tilde{J}\hat{A}\tilde{J} = A \otimes J$.

Thus,

$$M = \gamma_2^2 \cdot A \otimes J + (1 - \gamma_2^2)F,$$

and thus

$$\begin{aligned} \lambda(M) &\leq \gamma_2^2 \cdot \lambda(A \otimes J) + (1 - \gamma_2^2) \\ &\leq \gamma_2^2 \cdot (1 - \gamma_1) + (1 - \gamma_2^2) \\ &= 1 - \gamma_1\gamma_2^2, \end{aligned}$$

as desired. □

4.3.3 The Expander Construction

As a first attempt for constructing a family of expanders, we construct an infinite family $G_1, G_2, ...$ of graphs utilizing only the squaring and the zig-zag operations:

Construction 4.36 (Mildly Explicit Expanders). Let H be a $(D^4, D, 7/8)$-graph (e.g., as constructed in Problem 4.8), and define:

$$\begin{aligned} G_1 &= H^2 \\ G_{t+1} &= G_t^2 ⓩ H \end{aligned}$$

Proposition 4.37. For all t, G_t is a $(D^{4t}, D^2, 1/2)$-graph.

Proof. By induction on t.

Base Case: by the definition of H and Lemma 4.30, $G_1 = H^2$ is a $(D^4, D^2, 1-\lambda_0^2)$-graph and $\lambda_0^2 \leq 1/2$.

Induction Step: First note that $G_t^2 \textcircled{z} H$ is well-defined because $\deg(G_t^2) = \deg(G_t)^2 = (D^2)^2 = \#\text{nodes}(H)$. Then,

$$\begin{aligned} \deg(G_{t+1}) &= \deg(H)^2 = D^2 \\ \#\text{nodes}(G_{t+1}) &= \#\text{nodes}(G_t^2) \cdot \#\text{nodes}(H) = N_t \cdot D^4 = D^{4t} D^4 = D^{4(t+1)} \\ \lambda(G_{t+1}) &\leq \lambda(G_t)^2 + 2\lambda(H) \leq (1/2)^2 + 2 \cdot (1/8) = 1/2 \end{aligned}$$

□

Now, we recursively bound the time to compute neighbors in G_t. Actually, due to the way the G-step in the zig-zag product is defined, we bound the time to compute the *edge-rotation map* $(u, i) \mapsto (v, j)$, where the i'th edge leaving u equals the j'th edge entering v. Denote by $\text{time}(G_t)$ the time required for one evaluation of the edge-rotation map for G_t. This requires two evaluations of the edge-rotation map for G_{t-1} (the squaring requires two applications, while the zig-zag part does not increase the number of applications), plus time $\text{poly}(\log N_t)$ for manipulating strings of length $O(\log N_t)$. Therefore,

$$\begin{aligned} \text{time}(G_t) &= 2 \cdot \text{time}(G_{t-1}) + \text{poly}(\log N_t) \\ &= 2^t \cdot \text{poly}(\log N_t) \\ &= N_t^{\Theta(1)}, \end{aligned}$$

where the last equality holds because $N_t = D^{4t}$ for a constant D. Thus, this construction is only mildly explicit.

We remedy the above difficulty by using tensoring to make the sizes of the graphs grow more quickly:

Construction 4.38 (Fully Explicit Expanders). Let H be a $(D^8, D, 7/8)$-graph, and define:

$$\begin{aligned} G_1 &= H^2 \\ G_{t+1} &= (G_t \otimes G_t)^2 \textcircled{z} H \end{aligned}$$

In this family of graphs, the number of nodes grows doubly exponentially $N_t \approx c^{2^t}$, while the computation time grows only exponentially as before. Namely,

$$\text{time}(G_t) = 4^t \cdot \text{poly}(\log N_t) = \text{poly}(\log N_t).$$

We remark that the above family is rather sparse, i.e. the numbers in $\{N_t\}$ are far apart. To overcome this shortcoming, we can amend the above definition to have

$$G_t = (G_{\lceil t/2 \rceil} \otimes G_{\lfloor t/2 \rfloor})^2 \textcircled{z} H.$$

Now $N_t = D^{8t}$, so given a number N, we can find a graph G_t in the family whose size is at most $D^8 \cdot N = O(N)$. Moreover, the construction remains fully explicit because $\text{time}(G_t) = O(\text{time}(G_{\lceil t/2 \rceil}) + \text{time}(G_{\lfloor t/2 \rfloor})) = \text{poly}(t)$. Thus we have established:

Theorem 4.39. There is a constant $D \in \mathbb{N}$ such that for every $t \in \mathbb{N}$, there is a fully explicit expander graph G_t with degree D, spectral expansion $1/2$, and $N_t = D^{4t}$ nodes.

Consequently, the randomness-efficient error-reduction and averaging sampler based on expander walks can be made explicit:

Corollary 4.40. If a language L has a **BPP** algorithm with error probability at most $1/3$ that uses $m(n)$ random bits on inputs of length n, then for every polynomial $k(n)$, L has a **BPP** algorithm with error probability at most $2^{-k(n)}$ that uses $m(n) + O(k(n))$ random bits.

Corollary 4.41. There is an explicit averaging sampler achieving the parameters of Theorem 4.23.

4.3.4 Open Problems

As we have seen, spectral expanders such as those in Theorem 4.39 are also vertex expanders (Theorem 4.6 and Corollary 4.10) and edge expanders (Theorem 4.14), but these equivalences do not extend to optimizing the various expansion measures.

As mentioned in Section 4.3.1, there are known explicit constructions of optimal spectral expanders, namely Ramanujan graphs. However, unlike the expanders of Theorem 4.39, those constructions rely on deep results in number theory. The lack of a more elementary construction seems to signify a limitation in our understanding of expander graphs.

Open Problem 4.42. Give an explicit "combinatorial" construction of constant-degree expander graphs G with $\lambda(G) \leq 2\sqrt{D-1}/D$ (or even $\lambda(G) = O(\sqrt{D})$, where D is the degree.

For vertex expansion, it is known how to construct *bipartite* (or directed) expanders with constant left-degree (or out-degree) D and expansion $(1-\varepsilon) \cdot D$ for an arbitrarily small constant ε (see Chapter 6), but achieving the optimal expansion of $D - O(1)$ (cf., Theorem 4.4) or constructing undirected vertex expanders with high expansion remains open.

Open Problem 4.43. For an arbitrarily large constant D, give an explicit construction of bipartite $(\Omega(N), D-c)$ vertex expanders with N vertices on each side and left-degree D, where c is a universal constant independent of D.

Open Problem 4.44. For an arbitrarily small constant $\varepsilon > 0$, give an explicit construction of *undirected* $(\Omega(N), (1-\varepsilon)D)$ vertex expanders with N vertices and constant degree D that depends only on ε.

We remark that while Open Problem 4.43 refers to balanced bipartite graphs (i.e. ones with the same number of vertices on each side), the highly imbalanced case is also interesting and important. (See Problem 4.10 and Chapter 5.)

4.4 Undirected S-T Connectivity in Deterministic Logspace

Recall the Undirected S-T Connectivity problem: given an undirected graph G and two vertices s, t, decide whether there is a path from s to t. In Section 2.4, we saw that this problem can be solved in randomized logspace (**RL**). Here we will see how we can use expanders and the operations above to solve this problem in deterministic logspace (**L**).

The algorithm is based on the following two ideas:

- Undirected S-T Connectivity can be solved in logspace on constant-degree expander graphs. More precisely, it is easy on constant-degree graphs where every connected component is promised to be an expander (i.e. has spectral expansion bounded away from 1): we can try all paths of length $O(\log N)$ from s in logarithmic space; this works because expanders have logarithmic diameter. (See Problem 4.2.)
- The same operations we used to construct an infinite expander family above can also be used to turn *any* graph into an expander (in logarithmic space). Above, we started with a constant-sized expander and used various operations to build larger and larger expanders. There, the goal was to increase the size of the graph (which was accomplished by tensoring and/or zig-zag), while preserving the degree and the expansion (which was accomplished by zig-zag and squaring, which made up for losses in these parameters). Here, we want to improve the expansion (which will be accomplished by squaring), while preserving the degree (as will be handled by zig-zag) and ensuring the graph remains of polynomial size (so we will not use tensoring).

Specifically, the algorithm is as follows.

Algorithm 4.45 (Undirected S-T Connectivity in L).
Input: An undirected graph G with N edges and vertices s and t.

(1) Let H be a fixed $(D^4, D, 3/4)$ graph for some constant D.
(2) Reduce (G, s, t) to (G_0, s_0, t_0), where G_0 is a D^2-regular graph in which every connected component is nonbipartite and s_0 and t_0 are connected in G_0 iff s and t are connected in G.
(3) For $k = 1, \ldots, \ell = O(\log N)$, define:
   (a) Let $G_k = G_{k-1}^2 \textcircled{z} H$
   (b) Let s_k and t_k be any two vertices in the "clouds" of G_k corresponding to s_{k-1} and t_{k-1}, respectively. (Note that if s_k and t_k are connected in G_k, then s_{k-1} and t_{k-1} are connected in G_{k-1}.)
(4) Try all paths of length $O(\log N)$ in G_ℓ from s_ℓ and accept if any of them visit t_ℓ.

We will discuss how to implement this algorithm in logspace later, and first analyze its correctness. Let C_k be the connected component of G_k containing s_k. Observe that C_k is a connected component of $C_{k-1}^2 \textcircled{z} H$; below we will show that $C_{k-1}^2 \textcircled{z} H$ is connected and hence $C_k = C_{k-1}^2 \textcircled{z} H$. Since C_0 is undirected, connected, and nonbipartite, we have $\gamma(C_0) \geq 1/\text{poly}(N)$ by Theorem 2.53. We will argue that in each iteration the spectral gap increases by a constant factor, and thus after $O(\log N)$ iterations we have an expander.

By Lemma 4.30, we have

$$\gamma(C_{k-1}^2) \geq 2 \cdot \gamma(C_{k-1}) \cdot (1 - \gamma(C_{k-1})/2) \approx 2\gamma(C_k)$$

for small $\gamma(C_{k-1})$. By Theorem 4.35, we have

$$\begin{aligned} \gamma(C_{k-1}^2 \textcircled{z} H) &\geq \gamma(H)^2 \cdot \gamma(C_{k-1}^2) \\ &\geq \left(\frac{3}{4}\right)^2 \cdot 2 \cdot \gamma(C_{k-1}) \cdot (1 - \gamma(C_{k-1})/2) \\ &\geq \min\left\{\frac{35}{32} \cdot \gamma(C_{k-1}), \frac{1}{18}\right\} \end{aligned}$$

where the last inequality is obtained by considering whether $\gamma(C_{k-1}) \leq 1/18$ or $\gamma(C_{k-1}) > 1/18$. In particular, $C_{k-1}^2 \textcircled{z} H$ is connected, so we have $C_k = C_{k-1}^2 \textcircled{z} H$ and

$$\gamma(C_k) \geq \min\left\{\frac{35}{32} \cdot \gamma(C_{k-1}), \frac{1}{18}\right\}.$$

Thus, after $\ell = O(\log N)$ iterations, we must have $\gamma(C_\ell) \geq 1/18$. Moreover, observe that the number of vertices N_ℓ in G_ℓ is at most $N_0 \cdot (D^4)^\ell = \text{poly}(N)$, so considering paths of length $O(\log N)$ will suffice to decide s-t connectivity in G_ℓ.

To show that the algorithm can be implemented in logarithmic space, we argue that the edge-rotation map of each G_k can be computed with only $O(1)$ more space than the edge-rotation map of G_{k-1}, so that G_ℓ requires space $O(\log N) + O(\ell) = O(\log N)$. Since the inductive claim here refers to sublogarithmic differences of space (indeed $O(1)$ space) and sublogarithmic space is model-dependent (even keeping a pointer into the input requires logarithmic space), we will refer to a specific model of computation in establishing it. (The final result, that Undirected S-T Connectivity is in **L**, is, however, model-independent.) Formally, let space(G_k) denote the workspace needed to compute the edge-rotation map of G_ℓ on a multi-tape Turing machine with the following input/output conventions:

- Input Description:
  - Tape 1 (read-only): Contains the initial input graph G, with the head at the leftmost position of the tape.
  - Tape 2 (read-write): Contains the input pair (v, i), where v is a vertex of G_i and $i \in [D^2]$ is an index of the a neighbor on a *read-write* tape, with the head at the *rightmost* position of i. The rest of the tape may contain additional data.
  - Tapes 3+ (read-write): Blank worktapes with the head at the leftmost position.
- Output Description:
  - Tape 1: The head should be returned to the leftmost position.
  - Tape 2: In place of (v, i), it should contain the output (w, j) where w is the i'th neighbor of v and v is the j'th neighbor of w. The head should be at the rightmost position of j and the rest of the tape should remain unchanged from its state at the beginning of the computation.

– Tapes 3+ (read-write): Are returned to the blank state with the heads at the leftmost position.

With these conventions, it is not difficult to argue that space$(G_0) = O(\log N)$, and space$(G_k) =$ space$(G_{k-1}) + O(1)$. For the latter, we first argue that space$(G_{k-1}^2) =$ space$(G_{k-1}) + O(1)$, and then that space$(G_{k-1}^2 \circledz H) =$ space$(G_{k-1}^2) + O(1)$. For G_{k-1}^2, we are given a triple $(v, (i_1, i_2))$ on tape 2, with the head on the rightmost position of i_2, and both i_1 and i_2 are elements of $[D^2]$ (and thus of constant size). We move the head left to the rightmost position of i_1, compute the edge-rotation map of G_{k-1} on (v, i_1) so that tape 2 now contains (w, j_1, i_2). Then we swap j_1 and i_2, and run the edge-rotation map of G_{k-1} on (w, i_2) to get (w, j_2, j_1), and move the head to the rightmost position of j_1, completing the rotation. For $G_{k-1}^2 \circledz H$, we are given a tuple $((v, i), (a_1, a_2))$, where v is a vertex of G_{k-1}^2, i is a vertex of H (equivalently, an edge-label for G_{k-1}^2), and a_1, a_2 are edge labels for H. Evaluating the rotation map requires two evaluations of the rotation map for H (both of which are "constant-size" operations) and one evaluation of the rotation map of G_{k-1}^2.

Thus we have proven:

Theorem 4.46. Undirected S-T Connectivity is in **L**.

We remark that proving **RL** = **L** in general remains open. The best deterministic simulation known for **RL** is essentially $\mathbf{L}^{3/2} = \mathbf{DSPACE}(\log^{3/2} n)$, which makes beautiful use of known pseudorandom generators for logspace computation. (Unfortunately, we do not have space to cover this line of work in this survey.) Historically, improved derandomizations for Undirected S-T Connectivity have inspired improved derandomizations of **RL** (and vice-versa). Since Theorem 4.46 is still quite recent (2005), there is a good chance that we have not yet exhausted the ideas in it.

Open Problem 4.47. Show that $\mathbf{RL} \subset \mathbf{L}^c$ for some constant $c < 3/2$.

Another open problem is the construction of *universal traversal sequences* — fixed walks of polynomial length that are guaranteed to visit all vertices in any connected undirected regular graph of a given size. (See Example 3.8 and Open Problem 3.9.) Using the ideas from the algorithm above, it is possible to obtain logspace-constructible, polynomial-length universal traversal sequences for all regular graphs that are *consistently labelled* in the sense that no pair of distinct vertices have the same i'th neighbor for any $i \in [D]$. For general labellings, the best known universal traversal sequences are of length $N^{O(\log N)}$ (and are constructible in space $O(\log^2 N)$).

Open Problem 4.48 (Open Problem 3.9, restated). Give an explicit construction of universal traversal sequences of polynomial length for arbitrarily labelled undirected graphs (or even for an arbitrary labelling of the complete graph).

We remark that handling general labellings (for "pseudorandom walk generators" rather than universal traversal sequences) seems to be the main obstacle in extending the techniques of Theorem 4.46 to prove **RL** = **L**. (See the Chapter Notes and References.)

4.5 Exercises

~~Problem 4.1 (Bipartite vs. Nonbipartite Expanders).~~ ~~Show that constructing bipartite expanders~~ is equivalent to constructing (standard, nonbipartite) expanders. That is, show how given an explicit construction of one of the following, you can obtain an explicit construction of the other:

(1) D-regular $(\alpha N, A)$ expanders on N vertices for infinitely many N, where $\alpha > 0$, $A > 1$, and D are constants independent of N.
(2) D-regular (on both sides) $(\alpha N, A)$ bipartite expanders with N vertices on each side for infinitely many N, where $\alpha > 0$, $A > 1$, and D are constants independent of N.

(Your transformations need not preserve the constants.)

Problem 4.2 (More Combinatorial Consequences of Spectral Expansion). Let G be a graph on N vertices with spectral expansion $\gamma = 1 - \lambda$. Prove that:

(1) The *independence number* $\alpha(G)$ is at most $(\lambda/(1+\lambda))N$, where $\alpha(G)$ is defined to be the size of the largest independent set, i.e. subset S of vertices s.t. there are no edges with both endpoints in S.
(2) The *chromatic number* $\chi(G)$ is at least $(1+\lambda)/\lambda$, where $\chi(G)$ is defined to be the smallest number of colors for which the vertices of G can be colored s.t. all pairs of adjacent vertices have different colors.
(3) The *diameter* of G is $O(\log_{1/\lambda} N)$.

Recall that computing $\alpha(G)$ and $\chi(G)$ exactly are **NP**-complete problems. However, the above shows that for expanders, nontrivial bounds on these quantities can be computed in polynomial time.

Problem 4.3 (Limits on Vertex Expansion). This problem and the next one give limits on the vertex and spectral expansion that can be achieved as a function of the degree D. Both bounds are proved by relating the expansion of an arbitrary D-regular graph G by that of the infinite D-regular tree T_D (where every vertex has one parent and $D-1$ children), which is in some sense the "best possible" D-regular expander.

(1) Show that if a D-regular digraph G is a (K, A) expander, then T_D is a (K, A) expander.
(2) Show that for every $D \in \mathbb{N}$, there are infinitely many $K \in \mathbb{N}$ such that T_D is not a $(K, D - 1 + 2/K)$ expander.
(3) Deduce that for constant $D \in \mathbb{N}$ and $\alpha > 0$, if a D-regular, N-vertex digraph G is an $(\alpha N, A)$ vertex expander, then $A \leq D - 1 + o(1)$, where the $o(1)$ term vanishes as $N \to \infty$ (and D, α are held constant).

Problem 4.4 (Limits on Spectral Expansion). Let G be a D-regular undirected graph and T_D be the infinite D-regular tree (as in Problem 4.3). For a graph H and $\ell \in \mathbb{N}$, let $p_\ell(H)$ denote the probability that if we choose a random vertex v in H and do a random walk of length 2ℓ, we end back at vertex v.

(1) Show that $p_\ell(G) \geq p_\ell(T_D) \geq C_\ell \cdot (D-1)^\ell / D^{2\ell}$, where C_ℓ is the ℓ'th Catalan number, which equals the number of properly parenthesized strings in $\{(,)\}^{2\ell}$ — strings where no prefix has more)'s than ('s.
(2) Show that $N \cdot p_\ell(G) \leq 1 + (N-1) \cdot \lambda(G)^{2\ell}$. (Hint: use the fact that the trace of a matrix equals the sum of its eigenvalues.)
(3) Using the fact that $C_\ell = \binom{2\ell}{\ell}/(\ell+1)$, prove that

$$\lambda(G) \geq \frac{2\sqrt{D-1}}{D} - o(1),$$

where the $o(1)$ term vanishes as $N \to \infty$ (and D is held constant).

Problem 4.5 (Near-Optimal Sampling). (1) Describe an algorithm for Sampling that tosses $O(m + \log(1/\varepsilon) + \log(1/\delta))$ coins, makes $O((1/\varepsilon^2) \cdot \log(1/\delta))$ queries to a function $f : \{0,1\}^m \to [0,1]$, and estimates $\mu(f)$ to within $\pm\varepsilon$ with probability at least $1 - \delta$. (Hint: use expander walks to generate several sequences of coin tosses for the pairwise-independent averaging sampler, and compute the answer via a "median of averages".)
(2) Give an explicit (δ, ε) hitting sampler (see Problem 3.9) $\mathrm{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ that tosses $n = O(m + \log(1/\varepsilon) + \log(1/\delta))$ coins and generates $t = O((1/\varepsilon) \cdot \log(1/\delta))$ samples.

It turns out that these bounds on the randomness and query/sample complexities are each optimal up to constant factors (for most parameter settings of interest).

Problem 4.6 (Error Reduction For Free*). Show that if a problem has a **BPP** algorithm with constant error probability, then it has a **BPP** algorithm with error probability $1/n$ that uses *exactly* the same number of random bits.

Problem 4.7 (Vertex Expanders vs. Hitting Samplers). Given a bipartite multigraph with neighbor function $\Gamma : [N] \times [D] \to [M]$, we can obtain a sampler $\mathrm{Samp} : [N] \to [M]^D$ by setting $\mathrm{Samp}(x)_y = \Gamma(x, y)$. Conversely, every such sampler gives rise to a bipartite multigraph. Prove that Samp is a (δ, ε) hitting sampler if and only if Γ is an $(= K, A)$ vertex expander for $K = \delta N$ and $A = (1-\varepsilon)M/K$.

Thus, bipartite vertex expanders and hitting samplers are *equivalent*. However, the typical settings of parameters for the two objects are very different. For example, in vertex expanders, a primary goal is usually to maximize the expansion factor A, but KA may be significantly smaller

than M. In samplers, $AK = (1-\varepsilon)M$ is usually taken to be very close to M, but $A = (1-\varepsilon)M/\delta N$ may even be smaller than 1. Similarly, the most common setting of expanders takes $\delta = K/N$ to be a constant, whereas in samplers it is often thought of as vanishingly small.

Problem 4.8 (A "Constant-Sized" Expander). (1) Let $\mathbb{F}$ be a finite field. Consider a graph G with vertex set $\mathbb{F}^2$ and edge set $\{((a,b),(c,d)) : ac = b + d\}$. That is, we connect vertex (a,b) to all points on the line $y = ax - b$. Prove that G is $|\mathbb{F}|$-regular and $\lambda(G) \leq 1/\sqrt{|\mathbb{F}|}$. (Hint: consider G^2.)

(2) Show that if $|\mathbb{F}|$ is sufficiently large (but still constant), then by applying appropriate operations to G, we can obtain a base graph for the expander construction given in Section 4.3.3, i.e. a $(D^8, D, 7/8)$ graph for some constant D.

Problem 4.9 (The Replacement Product). Given a D_1-regular graph G_1 on N_1 vertices and a D_2-regular graph G_2 on D_1 vertices, consider the following graph $G_1 \textcircled{r} G_2$ on vertex set $[N_1] \times [D_1]$: vertex (u,i) is connected to (v,j) iff (a) $u = v$ and (i,j) is an edge in G_2, or (b) v is the i'th neighbor of u in G_1 and u is the j'th neighbor of v. That is, we "replace" each vertex v in G_1 with a copy of G_2, associating each edge incident to v with one vertex of G_2.

(1) Prove that there is a function g such that if G_1 has spectral expansion $\gamma_1 > 0$ and G_2 has spectral expansion $\gamma_2 > 0$ (and both graphs are undirected), then $G_1 \textcircled{r} G_2$ has spectral expansion $g(\gamma_1, \gamma_2, D_2) > 0$. (Hint: Note that $(G_1 \textcircled{r} G_2)^3$ has $G_1 \textcircled{z} G_2$ as a subgraph.)

(2) Show how to convert an explicit construction of constant-degree (spectral) expanders into an explicit construction of degree 3 (spectral) expanders.

(3) Without using Theorem 4.14, prove an analogue of Part 1 for edge expansion. That is, there is a function h such that if G_1 is an $(N_1/2, \varepsilon_1)$ edge expander and G_2 is a $(D_1/2, \varepsilon_2)$ edge expander, then $G_1 \textcircled{r} G_2$ is a $(N_1 D_1/2, h(\varepsilon_1, \varepsilon_2, D_2))$ edge expander, where $h(\varepsilon_1, \varepsilon_2, D_2) > 0$ if $\varepsilon_1, \varepsilon_2 > 0$. (Hint: given any set S of vertices of $G_1 \textcircled{r} G_2$, partition S into the clouds that are more than "half-full" and those that are not.)

(4) Prove that the functions $g(\gamma_1, \gamma_2, D_2)$ and $h(\varepsilon_1, \varepsilon_2, D_2)$ must depend on D_2, by showing that $G_1 \textcircled{r} G_2$ cannot be a $(N_1 D_1/2, \varepsilon)$ edge expander if $\varepsilon > 1/(D_2 + 1)$ and $N_1 \geq 2$.

Problem 4.10 (Unbalanced Vertex Expanders and Data Structures). Consider a $(K, (1-\varepsilon)D)$ bipartite vertex expander G with N left vertices, M right vertices, and left degree D.

(1) For a set S of left vertices, a $y \in N(S)$ is called a *unique neighbor* of S if y is incident to exactly one edge from S. Prove that every left-set S of size at most K has at least $(1-2\varepsilon)D|S|$ unique neighbors.

(2) For a set S of size at most $K/2$, prove that at most $|S|/2$ vertices outside S have at least δD neighbors in $N(S)$, for $\delta = O(\varepsilon)$.

Now we'll see a beautiful application of such expanders to data structures. Suppose we want to store a small subset S of a large universe $[N]$ such that we can test membership in S by probing just 1 bit of our data structure. A trivial way to achieve this is to store the characteristic vector of S, but this requires N bits of storage. The hashing-based data structures mentioned in Section 3.5.3 only require storing $O(|S|)$ words, each of $O(\log N)$ bits, but testing membership requires reading an entire word (rather than just one bit.)

Our data structure will consist of M bits, which we think of as a $\{0,1\}$-assignment to the right vertices of our expander. This assignment will have the following property.

Property Π: For all left vertices x, all but a $\delta = O(\varepsilon)$ fraction of the neighbors of x are assigned the value $\chi_S(x)$ (where $\chi_S(x) = 1$ iff $x \in S$).

(3) Show that if we store an assignment satisfying Property Π, then we can probabilistically test membership in S with error probability δ by reading just one bit of the data structure.

(4) Show that an assignment satisfying Property Π exists provided $|S| \leq K/2$. (Hint: first assign 1 to all of S's neighbors and 0 to all its nonneighbors, then try to correct the errors.)

It turns out that the needed expanders exist with $M = O(K \log N)$ (for any constant ε), so the size of this data structure matches the hashing-based scheme while admitting (randomized) 1-bit probes. However, note that such bipartite vertex expanders do *not* follow from explicit spectral expanders as given in Theorem 4.39, because the latter do not provide vertex expansion beyond $D/2$ nor do they yield highly imbalanced expanders (with $M \ll N$) as needed here. But in Chapter 5, we will see how to explicitly construct expanders that are quite good for this application (specifically, with $M = K^{1.01} \cdot \text{polylog}N$).

4.6 Chapter Notes and References

A detailed coverage of expander graphs and their applications is given by Hoory, Linial, and Wigderson [HLW].

The first papers on expander graphs appeared in conferences on telephone networks. Specifically, Pinsker [Pin] proved that random graphs are good expanders, and used these to demonstrate the existence of graphs called "concentrators." Bassalygo [Bas] improved Pinsker's results, in particular giving the general tradeoff between the degree D, expansion factor A, and set density α mentioned after Theorem 4.4. The first computer science application of expanders (and "superconcentrators") came in an approach by Valiant [Val] to proving circuit lower bounds. An early and striking algorithmic application was the $O(\log n)$-depth sorting network by Ajtai, Komlós, and Szemerédi [AKS2], which also illustrated the usefulness of expanders for derandomization. An exciting recent application of expanders is Dinur's new proof of the PCP Theorem [Din].

The fact that spectral expansion implies vertex expansion and edge expansion was shown by Tanner [Tan] (for vertex expansion) and Alon and Milman [AM] (for edge expansion). The converses are discrete analogues of Cheeger's Inequality for Riemannian manifolds [Che1], and various forms of these were proven by Alon [Alo1] (for vertex expansion) and Jerrum and Sinclair [JS] (for edge expansion in undirected graphs and, more generally, conductance in reversible Markov chains) and

Mihail [Mih] (for edge expansion in regular digraphs and conductance in non-reversible Markov chains).

The "Ramanujan" upper bound on spectral expansion given by Theorem 4.11 was proven by Alon and Boppana (see [Alo1, Nil]). Theorem 4.12, stating that random graphs are asymptotically Ramanujan, was conjectured by Alon [Alo1], but was only proven recently by Friedman [Fri]. Kahale [Kah] proved that Ramanujan graphs have vertex expansion roughly $D/2$ for small sets.

Forms of the Expander Mixing Lemma date back to Alon and Chung [AC2], who considered the number of edges between a set and its complement (i.e. $T = V \setminus S$). The converse to the Expander Mixing Lemma (Theorem 4.16) is due to Bilu and Linial [BL]. For more on quasirandomness, see [CGW, AS1] for the case of dense graphs and [CG2, CG3] for sparse graphs.

The sampling properties of random walks on expanders were analyzed in a series of works starting with Ajtai, Komlós, and Szemerédi [AKS3]. The hitting bound of Theorem 4.17 is due to Kahale [Kah], and the Chernoff Bound for expander walks (cf., Theorem 4.22) is due to Gillman [Gil3]. Our proof of the Chernoff Bound is inspired by that of Healy [Hea], who also provides some other variants and generalizations. The **RP** version of Problem 4.6 is due to Karp, Pippenger, and Sipser [KPS], who initiated the study of randomness-efficient error reduction of randomized algorithms. It was generalized to **BPP** in [CW1]. The equivalence of hitting samplers and bipartite vertex expanders from Problem 4.7 is due to Sipser **??**. Problem 4.5 is due to Bellare, Goldreich, and Goldwasser [BGG]; matching lower bounds for sampling were given by Canetti, Even, and Goldreich [CEG]. Open Problem 4.24 was posed by Bellare and Rompel [BR].

Construction 4.25 is due to Margulis [Mar1], and was the first explicit construction of constant-degree expanders. Gabber and Galil [GG] (see also [JM]) gave a much more elementary proof of expansion for similar expanders, which also provided a specific bound on the spectral expansion (unlike Margulis' proof). Construction 4.26 is variant of a construction of Lubotzky, Phillips, and Sarnak. (See [Lub, Thm. 4.42], from which the expansion of Construction 4.26 can be deduced.) Ramanujan graphs (Construction 4.27) were constructed independently by Lubotzky, Phillips, and Sarnak [LPS] and Margulis [Mar2]. For more on Ramanujan graphs and the mathematical machinery that goes into their analysis, see the books [Sar, Lub, DSV].

The zig-zag product and the expander constructions of Section 4.3.3 are due to Reingold, Vadhan, and Wigderson [RVW]. Our analysis of the zig-zag product is from [RTV], which in turn builds on [RV], who used matrix decomposition (Lemma 4.19) for analyzing other graph products. Earlier uses of graph products in constructing expanders include the use of the tensor product in [Tan]. Problem 4.9, on the replacement product, is from [RVW, RTV], and can be used in place of the zig-zag product in both the expander constructions and the UNDIRECTED S-T CONNECTIVITY algorithm (Algorithm 4.45). Independently of [RVW], Martin and Randall [MR] proved a "decomposition theorem" for Markov chains that implies a better bound on the spectral expansion of the replacement product.

There has been substantial progress on giving a combinatorial construction of Ramanujan graphs (Open Problem 4.42). Bilu and Linial [BL] give a mildly explicit construction achieving $\lambda(G) = \tilde{O}(\sqrt{D})$, Ben-Aroya and Ta-Shma [BT] give a fully explicit construction achieving $\lambda(G) = D^{1/2+o(1)}$, and Spielman et al. [BSS] give a mildly explicit construction of a *weighted* graph achieving $\lambda(G) = O(\sqrt{D})$.

Constant-degree bipartite expanders with expansion $(1-\varepsilon)\cdot D$ have been constructed by Ca-

palbo et al. [CRVW]. Alon and Capalbo [AC1] have made progress on Open Problem 4.44 by giving an explicit construction of undirected constant-degree "unique-neighbor" expanders (see Problem 4.10).

The deterministic logspace algorithm for Undirected S-T Connectivity (Algorithm 4.45) is due to Reingold [Rei]. The result that $\mathbf{RL} \subset \mathbf{L}^{3/2}$ is due to Saks and Zhou [SZ], with an important ingredient being Nisan's pseudorandom generator for space-bounded computation [Nis]. This was slightly improved by Armoni [Arm], who showed that $\mathbf{RL} \subset \mathbf{DSPACE}((\log^{3/2} n)/\sqrt{\log\log n}))$, which remains the best known derandomization of **RL**.

Based on Algorithm 4.45, explicit polynomial-length universal traversal sequences for "consistently labelled" regular digraphs, as well as "pseudorandom walk generators" for such graphs, were constructed in [Rei, RTV]. (See also [RV].) In [RTV], it is shown that pseudorandom walk generators for arbitrarily labelled regular digraphs would imply $\mathbf{RL} = \mathbf{L}$. The best known explicit construction of a full-fledged universal traversal sequence is due to Nisan [Nis], has length $n^{O(\log n)}$, and can be constructed in time $n^{O(\log n)}$ and space $O(\log^2 n)$.

Problem 4.8, Part 1 is a variant of a construction of Alon [Alo2]; Part 2 is from [RVW]. The results of Problem 4.2 are from [Hof, Lov2, AM, Chu]. The result of Problem 4.10, on bit-probe data structures for set membership, is due to Buhrman, Miltersen, Radhakrishnan, and Venkatesan [BMRV].

5

List-Decodable Codes

The field of *coding theory* is motivated by the problem of communicating reliably over noisy channels — where the data sent over the channel may come out corrupted on the other end, but we nevertheless want the receiver to be able to correct the errors and recover the original message. There is a vast literature studying aspects of this problem from the perspectives of electrical engineering (communications and information theory), computer science (algorithms and complexity), and mathematics (combinatorics and algebra). In this survey, we are interested in codes as "pseudorandom objects." In particular, a generalization of the notion of an error-correcting code yields a framework that we will use to unify all of the main pseudorandom objects covered in this survey (averaging samplers, expander graphs, randomness extractors, list-decodable codes, pseudorandom generators).

5.1 Definitions and Existence

5.1.1 Definition.

The approach to communicating over a noisy channel is to restrict the data we send to be from a certain set of strings that can be easily disambiguated (even after being corrupted).

Definition 5.1. A *q-ary code* is a multiset[1] $\mathcal{C} \subseteq \Sigma^{\hat{n}}$, where Σ is an *alphabet* of size q. Elements of $\mathcal{C}$ are called *codewords*. We definet he following key parameters:

- $\hat{n}$ is the *block length*.
- $n = \log_2 |\mathcal{C}|$ is the *message length*.
- $\rho = n/(\hat{n} \cdot \log |\Sigma|)$ is the *(relative) rate* of the code.

An *encoding function* for $\mathcal{C}$ is an injective mapping $\mathrm{Enc}\colon \{1, \ldots, |\mathcal{C}|\} \to \mathcal{C}$. Given such an encod-

[1] Traditionally, codes are defined to be sets rather than multisets. However, the generality afforded by multisets will allow the connections we see later to be stated more cleanly. In the case $\mathcal{C}$ is a multiset, the condition that a mapping $\mathrm{Enc} : \{1, \ldots, N\} \to \mathcal{C}$ is injective should be replaced with the constraint that for every $c \in \mathcal{C}$, $|\mathrm{Enc}^{-1}(c)|$ is no larger than the multiplicity of c in $\mathcal{C}$.

ing function, we view the elements of $\{1, \ldots, |\mathcal{C}|\}$ as *messages*. When $n = \log |\mathcal{C}|$ is an integer, we often think of messages as strings in $\{0,1\}^n$.

We view $\mathcal{C}$ and Enc as being essentially the same object (with Enc merely providing a "labelling" of codewords), with the former being useful for studying the combinatorics of codes and the latter for algorithmic purposes.

We remark that our notation differs from the standard notation in coding theory in several ways. Typically in coding theory, the input alphabet is taken to be the same as the output alphabet (rather than $\{0,1\}$ and Σ, respectively), the blocklength is denoted n, and the message length (over Σ) is denoted k and is referred to as the rate.

So far, we haven't talked at all about the error-correcting properties of codes. Here we need to specify two things: the model of errors (as introduced by the noisy channel) and the notion of a successful recovery.

For the errors, the main distinction is between *random errors* and *worst-case errors*. For random errors, one needs to specify a stochastic model of the channel. The most basic one is the *binary symmetric channel* (over alphabet $\Sigma = \{0,1\}$), where each bit is flipped independently with probability δ. People also study more complex channel models, but as usual with stochastic models, there is always the question of how well the theoretical model correctly captures the real-life distribution of errors. We, however, will focus on *worst-case errors*, where we simply assume that fewer than a δ fraction of symbols have been changed. That is, when we send a codeword $c \in \Sigma^{\hat{n}}$ over the channel, the received word $r \in \Sigma^{\hat{n}}$ differs from c in fewer than $\delta\hat{n}$ places. Equivalently, c are r are close in Hamming distance:

Definition 5.2 (Hamming distance). For two strings $x, y \in \Sigma^{\hat{n}}$, their *(relative) Hamming distance* $d_H(x,y)$ equals $\Pr_i[x_i \neq y_i]$. The *agreement* is defined to be $\mathrm{agr}(x,y) = 1 - d_H(x,y)$.

For a string $x \in \Sigma^{\hat{n}}$ and $\delta \in [0,1]$, the *(open) Hamming ball* of radius δ around x is the set $B(x,\delta)$ of strings $y \in \Sigma^{\hat{n}}$ such that $d_H(x,y) < \delta$.

For the notion of a successful recovery, the traditional model requires that we can uniquely decode the message from the received word (in the case of random errors, this need only hold with high probability). Our main focus will be on a more relaxed notion which allows us to produce a small list of candidate messages. As we will see, the advantage of such list-decoding is that it allows us to correct a larger fraction of errors.

Definition 5.3. Let $\mathrm{Enc}\colon \{0,1\}^n \to \Sigma^{\hat{n}}$ be an encoding algorithm for a code $\mathcal{C}$. A *δ-decoding* algorithm for Enc is a function $\mathrm{Dec} : \Sigma^{\hat{n}} \to \{0,1\}^n$ such that for every $m \in \{0,1\}^n$ and $r \in \Sigma^{\hat{n}}$ satisfying $d_H(\mathrm{Enc}(m), r) < \delta$, we have $\mathrm{Dec}(r) = m$. If such a function Dec exists, we call the code *δ-decodable*.

A *(δ, L)-list-decoding* algorithm for Enc is a function $\mathrm{Dec} : \Sigma^{\hat{n}} \to (\{0,1\}^n)^L$ such that for every $m \in \{0,1\}^n$ and $r \in \Sigma^{\hat{n}}$ satisfying $d_H(\mathrm{Enc}(m), r) < \delta$, we have $m \in \mathrm{Dec}(r)$. If such a function *Dec* exists, we call the code *(δ, L)-list-decodable*.

Note that a δ-decoding algorithm is the same as a $(\delta, 1)$-list-decoding algorithm. It is not hard to see that, if we do not care about computational efficiency, the existence of such decoding algorithms depends only on the combinatorics of the set of codewords.

Definition 5.4. The *(relative) minimum distance* of a code $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ equals $\min_{x \neq y \in \mathcal{C}} d_H(x, y)$.[2]

Proposition 5.5. Let $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ be a code with any encoding function Enc.

(1) For $\delta\hat{n} \in \mathbb{N}$, Enc is δ-decodable iff its minimum distance is at least $2\delta - 1/\hat{n}$.
(2) Enc is (δ, L)-list-decodable iff for every $r \in \Sigma^{\hat{n}}$, we have $|B(r, \delta) \cap \mathcal{C}| \leq L$.

Proof. Item 2 follows readily from the definitions.

For Item 1, first note that Enc is δ-decodable iff there is no received word $r \in \Sigma^{\hat{n}}$ at distance less than δ from two codewords c_1, c_2. (Such an r should decode to both c_1 and c_2, which is impossible for a unique decoder.) If the code has minimum distance at least $2\delta - 1/\hat{n}$, then c_1 and c_2 disagree in at least $2\delta\hat{n} - 1$ positions, which implies that r disagrees with one of them in at least $\delta\hat{n}$ positions. (Recall that $\delta\hat{n}$ is an integer by hypothesis.) Conversely, if the code has minimum distance smaller than $2\delta - 1/\hat{n}$, then there are two codewords that disagree in at most $2\delta\hat{n} - 2$ positions, and we can construct r that disagrees with each in at most $\delta\hat{n} - 1$ positions. □

Because of the factor of 2 in Item 1, unique decoding is only possible at distances up to $1/2$, whereas we will see that list-decoding is possible at distances approaching (with small lists).

The main goals in constructing codes are to have infinite families of codes (e.g. for every message length n) in which we:

- Maximize the fraction δ of errors correctible (e.g. constant independent of n and $\hat{n}$).
- Maximize the rate ρ (e.g. a constant independent of n and $\hat{n}$).
- Minimize the alphabet size q (e.g. a constant, ideally $q = 2$).
- Keep the list size L relatively small (e.g. a constant or $\text{poly}(n)$).
- Have computationally efficient encoding and decoding algorithms.

In particular, coding theorists are very interested in obtaining the optimal tradeoff between the constants δ and ρ with efficiently encodable and decodable codes.

5.1.2 Existential Bounds

Like expanders, the existence of very good codes can be shown using the probabilistic method. The bounds will be stated in terms of the q-ary entropy functions, so we begin by defining those.

Definition 5.6. For $q, \hat{n} \in \mathbb{N}$ and $\delta \in [0, 1]$, we define $H_q(\delta, \hat{n}) \in [0, 1]$ to be such that $|B(x, \delta)| = q^{H_q(\delta,\hat{n}) \cdot \hat{n}}$ for $x \in \Sigma^{\hat{n}}$, where Σ is an alphabet of size q.

We also define the *q-ary entropy function* $H_q(\delta) = \delta \cdot \log_q((q-1)/\delta) + (1-\delta) \cdot \log_q(1/(1-\delta))$.

The reason we use similar notation for $H_q(\delta, \hat{n})$ and $H_q(\delta)$ is Part 1 of the following:

[2] If any codeword appears in $\mathcal{C}$ with multiplicity greater than 1, then the minimum distance is defined to be zero.

Proposition 5.7. For every $q \in \mathbb{N}$, $\delta \in [0,1]$, $\varepsilon \in [0,1/2]$,

(1) $\lim_{\hat{n}\to\infty} H_q(\delta, \hat{n}) = H_q(\delta)$.
(2) $H_q(\delta) \le H_2(\delta)/\log q + \delta$.
(3) $H_2(1/2 - \varepsilon) = \Theta(\varepsilon^2)$.

Now we state the bounds for random error-correcting codes.

Theorem 5.8. (1) For all $\hat{n}, q \in \mathbb{N}$ and $\delta \in (0, 1 - 1/q)$, there exists a q-ary code of block length $\hat{n}$, minimum distance at least δ, and rate at least $\rho = 1 - H_q(\delta, \hat{n})$.
(2) For all all integers $\hat{n}, q, L \in \mathbb{N}$ and $\delta \in (0, 1 - 1/q)$, there exists a (δ, L)-list-decodable q-ary code of block length $\hat{n}$ and rate at least $\rho = 1 - H_q(\delta, \hat{n}) - 1/(L+1)$.

Proof. (1) Pick the codewords $c_1, \ldots, c_N$ in sequence ensuring that c_i is at distance at least δ from $c_1, \ldots, c_{i-1}$. The union of the Hamming balls of radius δ around $c_1, \ldots, c_{i-1}$ contains at most $(i-1) \cdot q^{H_q(\delta,\hat{n})\cdot\hat{n}} < (N-1) \cdot q^{H_q(\delta,\hat{n})\cdot\hat{n}}$, so there is always a choice of c_i outside these balls if we take $N = \lceil q^{(1-H_q(\delta,\hat{n}))\cdot\hat{n}} \rceil$.

(2) We use the probabilistic method. Choose the N codewords randomly and independently from $\Sigma^{\hat{n}}$. The probability that there is a Hamming Ball of radius δ containing at least $L+1$ codewords is at most

$$q^{\hat{n}} \cdot \binom{N}{L+1} \cdot \left(\frac{q^{H_q(\delta,\hat{n})\cdot\hat{n}}}{q^{\hat{n}}}\right)^{L+1} \le \left(\frac{N-1}{q^{(1-H_q(\delta,\hat{n})-1/(L+1))\cdot\hat{n}}}\right)^{L+1},$$

which is less than 1 if we take $N = \lceil q^{(1-H_q(\delta,\hat{n})-1/(L+1))\cdot\hat{n}} \rceil$.

□

Note that while the rate bounds are essentially the same for achieving minimum distance and the list-decoding radius δ (as we take large list size), recall that minimum distance δ only corresponds to unique decoding up to radius roughly $\delta/2$. The bound for list-decoding is known to be tight up to the dependence on L (Problem 5.1), while the bound on minimum distance is not tight in general. Indeed, there are families "algebraic-geometric" codes with constant alphabet size q, constant minimum distance $\delta > 0$, and constant rate $\rho > 0$ where $\rho > 1 - H_q(\delta, \hat{n})$ for sufficiently large $\hat{n}$. (Thus, this is a rare counterexample to the phenomenon "random is best".) Identifying the best tradeoff between rate and minimum distance, even for binary codes, is a long-standing open problem in coding theory.

Open Problem 5.9. For each constant $\delta \in (0,1)$, identify the largest $\rho > 0$ such that for every $\varepsilon > 0$, there exists an infinite family of codes $\mathcal{C}_{\hat{n}} \subseteq \{0,1\}^{\hat{n}}$ of rate at least $\rho - \varepsilon$ and minimum distance at least δ.

Let's look at some special cases of the parameters in the above theorem. For binary codes ($q = 2$), we will be most interested in the case $\delta \to 1/2$, which corresponds to correcting the

maximum possible fraction of errors for binary codes. (No nontrivial decoding is possible for binary codes at distance greater than $1/2$, since a completely random received word will be at distance roughly $1/2$ with most codewords.) In this case, Proposition 5.7 tells us that the rate approaches $1 - H_2(1/2 - \varepsilon) = \Theta(\varepsilon^2)$, i.e. the blocklength is $\hat{n} = \Theta(n/\varepsilon^2)$ (for list size $L = \Theta(1/\varepsilon^2)$). For large alphabets q, Proposition 5.7 tells us that the rate approaches $1 - \delta$ as q grows. We will be most interested in the case $\delta = 1 - \varepsilon$ for small ε, so we can correct a $\delta = 1 - \varepsilon$ fraction of errors with a rate arbitrarily close to ε. For example, we can achieve rate of $\rho = .99\varepsilon$, a list size of $L = O(1/\varepsilon)$ and an alphabet of size $\text{poly}(1/\varepsilon)$. More generally, it is possible to achieve rate $\rho = (1 + \gamma)\varepsilon$ with an alphabet size of $q = (1/\varepsilon)^{O(1/\gamma)}$.

While we are primarily interested in list-decodable codes, minimum distance is often easier to bound. The following allows us to translate bounds on minimum distance into bounds on list-decodability.

Proposition 5.10 (Johnson Bound). (1) If $\mathcal{C}$ has minimum distance $1 - \varepsilon$, then it is $(1 - O(\sqrt{\varepsilon}), O(1/\sqrt{\varepsilon}))$-list-decodable.

(2) If a *binary* code $\mathcal{C}$ has minimum distance $1/2 - \varepsilon$, then it is $(1/2 - O(\sqrt{\varepsilon}), O(1/\varepsilon))$-list-decodable.

Proof. We prove Item 1, and leave Item 2 as an exercise in the next chapter (Problem **??**). The proof is by inclusion-exclusion. Suppose for contradiction that there are codewords $c_1, \ldots, c_s$ at distance less than $1 - \varepsilon'$ from some $r \in \Sigma^{\hat{n}}$, for $\varepsilon' = 2\sqrt{\varepsilon}$ and $s = \lceil 2/\varepsilon' \rceil$. Then:

$$\begin{aligned} 1 &\geq \text{fraction of positions where } r \text{ agrees with some } c_i \\ &\geq \sum_i \text{agr}(r, c_i) - \sum_{1 \leq i < j \leq s} \text{agr}(c_i, c_j) \\ &> s\varepsilon' - \binom{s}{2} \cdot \varepsilon \\ &\geq 2 - 1 = 1 \end{aligned}$$

where the last inequality is by our setting of parameters. Contradiction. □

Note the quadratic loss in the distance parameter. This means that optimal codes with respect to minimum distance are not necessarily optimal with respect to list-decoding. Nevertheless, if we do not care about the exact tradeoff between the rate and the decoding radius, the above can yield codes where the decoding radius is as large as possible (approaching 1 for large alphabets and $1/2$ for binary alphabets).

5.1.3 Explicit Codes

As usual, most applications of error-correcting codes (in particular the original motivating one) require computationally efficient encoding and decoding. For now, we focus on only the efficiency of encoding.

Definition 5.11. A code $\text{Enc} : \{0,1\}^n \to \Sigma^{\hat{n}}$ is *(fully) explicit* if given a message $m \in \{0,1\}^n$ and an index $i \in \hat{n}$, the i'th symbol of $\text{Enc}(m)$ can be computed in time $\text{poly}(n, \log \hat{n}, \log |\Sigma|)$.

The reason we talk about computing individual symbols of the codeword rather than the entire codeword is to have a meaningful definition even for codes where the blocklength $\hat{n}$ is superpolynomial in the message length n. One can also consider weaker notions of explicitness, where we simply require that the entire codeword $\mathrm{Enc}(m)$ can be computed in time $\mathrm{poly}(\hat{n}, \log|\Sigma|)$.

The constructions of codes that we describe will involve arithmetic over finite fields. See Remark 3.25 for the complexity of constructing finite fields and carrying out arithmetic in such fields .

In describing the explicit constructions below, it will be convenient to think of the codewords as functions $c : [\hat{n}] \to \Sigma$ rather than as strings in $\Sigma^{\hat{n}}$.

Construction 5.12 (Hadamard Code). For $n \in \mathbb{N}$, the *(binary) Hadamard code* of message length n is the binary code of blocklength $\hat{n} = 2^n$ consisting of all functions $c : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that are *linear* (modulo 2).

Proposition 5.13. The Hadamard code:

(1) is explicit with respect to the encoding function that takes a message $m \in \mathbb{Z}_2^n$ to the linear function c_m defined by $c_m(x) = \sum_i m_i x_i \bmod 2$.
(2) has minimum distance $1/2$, and
(3) is $(1/2 - \varepsilon, O(1/\varepsilon^2))$ list-decodable for every $\varepsilon > 0$.

Proof. Explicitness is clear by inspection. The minimum distance follows from the fact that for every two distinct linear functions $c, c' : \mathbb{Z}_2^m \to \mathbb{Z}_2$, $\Pr_x[c(x) = c'(x)] = \Pr_x[(c - c')(x) = 0] = 1/2$. The list-decodability follows from the Johnson Bound. □

The advantages of the Hadamard code are its small alphabet (binary) and optimal distance $(1/2)$, but unfortunately its rate is exponentially small ($\rho = n/2^n$). By increasing both the field size and degree, we can obtain complementary properties

---

Construction 5.14 (Reed–Solomon Codes). For a prime power q and $d \in \mathbb{N}$, the q-ary *Reed–Solomon code* of degree d is the code of blocklength $\hat{n} = q$ and message length $n = (d+1) \cdot \log q$ consisting of all polynomials $p : \mathbb{F}_q \to \mathbb{F}_q$ of degree at most d.

Proposition 5.15. The q-ary Reed–Solomon Code of degree d:

(1) is explicit with respect to the encoding function that takes a vector of coefficients $m \in \mathbb{F}_q^{d+1}$ to the polynomial p_m defined by $p_m(x) = \sum_{i=0}^{d} m_i x^i$ (assuming we have a description of the field $\mathbb{F}_q$).
(2) has minimum distance $\delta = 1 - d/q$, and
(3) is $(1 - O(\sqrt{d/q}), O(\sqrt{q/d}))$ list-decodable.

Proof. Again explicitness follows by inspection. The minimum distance follows from the fact that two distinct polynomials of degree at most d agree in at most d points (else their difference would have more than d roots). The list-decodability follows from the Johnson Bound. □

Note that by setting $q = O(d)$, Reed-Solomon codes simultaneously achieve constant rate and constant distance, the only disadvantage being that the alphabet is of nonconstant size (namely $q = \hat{n} > n$.)

Another useful setting of parameters for Reed-Solomon codes in complexity theory is $q = \text{poly}(d)$, which gives polynomial rate ($\hat{n} = \text{poly}(n)$) and distance tending to 1 polynomially fast ($\delta = 1 - 1/\text{poly}(n)$).

The following codes "interpolate" between Hadamard and Reed-Solomon codes, by allowing the number of variables, the degree, and field size all to vary.

Construction 5.16 (Reed–Muller Codes). For a prime power q and $d, m \in \mathbb{N}$, the q-ary *Reed–Muller code* of degree d and dimension t is the code of blocklength $\hat{n} = q^m$ and message length $n = \binom{m+d}{d} \cdot \log q$ consisting of all polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ of (total) degree at most d.

Proposition 5.17. The q-ary Reed–Muller Code of degree d and dimension m:

(1) is explicit with respect to the encoding function that takes a vector of coefficients $v \in \mathbb{F}_q^{\binom{m+d}{d}}$ to the corresponding polynomial p_v.
(2) has minimum distance $\delta \geq 1 - d/q$, and
(3) is $(1 - O(\sqrt{d/q}), O(\sqrt{q/d}))$ list-decodable.

Proof. Same as for Reed–Solomon Codes, except we use the Schwartz-Zippel Lemma (Lemma 2.4) to deduce the minimum distance. □

Note that Reed–Solomon Codes are simply Reed–Muller codes of dimension $m = 1$, and Hadamard codes are essentially Reed–Muller codes of degree $d = 1$ and alphabet size $q = 2$ (except that the Reed–Muller code also contains affine linear functions).

5.2 List-Decoding Algorithms

In this section, we will describe efficient list-decoding algorithms for the Reed–Solomon code and variants. It will be convenient to work with the following notation:

Definition 5.18. Let $\mathcal{C}$ be a code with encoding function $\text{Enc} : \{1, \ldots, N\} \to \Sigma^{\hat{n}}$. For $r \in \Sigma^{\hat{n}}$, define $\text{LIST}(r, \varepsilon) = \{m : \text{agr}(\text{Enc}(m), r) > \varepsilon\}$.

Then the task of $(1 - \varepsilon)$ list-decoding (according to Definition 5.3) is equivalent to producing the elements of $\text{LIST}(r, \varepsilon)$ given $r \in \Sigma^{\hat{n}}$. In this section, we will see algorithms that do this in time polynomial in the bit-length of r, i.e. time $\text{poly}(\hat{n}, \log |\Sigma|)$.

5.2.1 Review of Algebra

The list-decoding algorithms will require some additional algebraic facts and notation:

- For every field $\mathbb{F}$, $\mathbb{F}[X_1, \ldots, X_n]$ is the integral domain consisting of formal polynomials $Q(X_1, \ldots, X_n)$ with coefficients in $\mathbb{F}$, where addition and multiplication of polynomials is defined in the usual way.
- A nonzero polynomial $Q(X_1, \ldots, X_n)$ is *irreducible* if we cannot write $Q = RS$ where R, S are nonconstant polynomials. For a finite field $\mathbb{F}_q$ of characteristic p and $d \in \mathbb{N}$, a univariate irreducible polynomial of degree d over $\mathbb{F}_q$ can be found in deterministically in time $\text{poly}(p, \log q, d)$.
- $\mathbb{F}[X_1, \ldots, X_n]$ is a *unique factorization domain*. That is, every nonzero polynomial Q can be factored as $Q = Q_1 Q_2 \cdots Q_m$, where each Q_i is irreducible and this factorization is unique up to reordering and multiplication by constants from $\mathbb{F}$. Given the description of a finite field $\mathbb{F}_{p^k}$ and the polynomial Q, this factorization can be done probabilistically in time $\text{poly}(\log p, k, |Q|)$ and deterministically in time $\text{poly}(p, k, |Q|)$.
- For a nonzero polynomial $Q(Y, Z) \in \mathbb{F}[Y, Z]$ and $f(Y) \in \mathbb{F}[Y]$, if $Q(Y, f(Y)) = 0$, then $Z - f(Y)$ is one of the irreducible factors of $Q(Y, Z)$ (and thus $f(Y)$ can be found in polynomial time). This is analogous to the fact that if $c \in \mathbb{Z}$ is a root of an integer polynomial $Q(Z)$, then $Z - c$ is one of the factors of Q (and can be proven in the same way, by long division).

5.2.2 List-Decoding Reed-Solomon Codes

Theorem 5.19. There is a polynomial-time $(1 - \varepsilon)$ list-decoding algorithm for the q-ary Reed-Solomon code of degree d, for $\varepsilon = 2\sqrt{d/q}$. That is, given a function $r : \mathbb{F}_q \to \mathbb{F}_q$ and $d \in \mathbb{N}$, all polynomials of degree at most d that agree with r on more than $\varepsilon q = 2\sqrt{dq}$ inputs can be found in polynomial time.

In fact the constant of 2 can be improved to 1, matching the combinatorial list-decoding radius for Reed–Solomon codes given by an optimized form of the Johnson Bound. See Problem 5.6.

Proof. We are given a received word $r : \mathbb{F}_q \to \mathbb{F}_q$, and want to find all elements of $\text{LIST}(r, \varepsilon)$ for $\varepsilon = 2\sqrt{d/q}$.

Step 1: Find a low-degree Q "explaining" r. Specifically, $Q(Y, Z)$ will be a nonzero bivariate polynomial of degree at most d_Y in its first variable Y and d_Z in its second variable, and will satisfy $Q(y, r(y)) = 0$ for all $y \in \mathbb{F}_q$. Each such y imposes a linear constraint on the $(d_Y + 1)(d_Z + 1)$ coefficients of Q. Thus, this system has a nonzero solution provided $(d_Y + 1)(d_Z + 1) > q$, and it can be found in polynomial time by linear algebra (over $\mathbb{F}_q$).

Step 2: Argue that each $f(Y) \in \text{LIST}(r, \varepsilon)$ is a "root" of Q. Specifically, it will be the case that $Q(Y, f(Y)) = 0$ for each $f \in \text{LIST}(r, \varepsilon)$. The reason is that $Q(Y, f(Y))$ is a univariate polynomial of degree at most $d_Y + d \cdot d_Z$, and has more than εq zeroes (one for each place that f and r agree). Thus, we can conclude $Q(Y, f(Y)) = 0$ provided $\varepsilon q \geq d_Y + d \cdot d_Z$. Then we can

enumerate all of the elements of LIST(r) by factoring $Q(Y, Z)$ and taking all the factors of the form $Z - f(Y)$.

For this algorithm to work, the two conditions we need to satisfy are

$$(d_Y + 1)(d_Z + 1) > q,$$

and

$$\varepsilon q \geq d_Y + d \cdot d_Z.$$

These conditions can be satisfied by setting $d_Y = \lfloor \varepsilon q/2 \rfloor$, $d_Z = \lfloor \varepsilon q/(2d) \rfloor$, and $\varepsilon = 2\sqrt{d/q}$. □

Note that the rate of Reed-Solomon codes is $\rho = (d+1)/q = \Theta(\varepsilon^2)$. The alphabet size is $q = \tilde{\Omega}(n/\rho) = \tilde{\Omega}(n/\varepsilon^2)$. In contrast, the random codes of Theorem 5.8 achieve $\rho \approx \varepsilon$ and $q = \text{poly}(1/\varepsilon)$. It is not known whether the $\varepsilon = \sqrt{d/q}$ bound on the list-decodability of Reed–Solomon codes can be improved, even with inefficient decoding.

Open Problem 5.20. Do there exist constants $\varepsilon, \rho \in (0, 1)$ with $\varepsilon < \sqrt{\rho}$ and an infinite sequence of prime powers q such that the q-ary Reed–Solomon code of degree $d = \lfloor \rho q \rfloor$ is $(1 - \varepsilon, \text{poly}(q))$-list-decodable?

5.2.3 Parvaresh–Vardy Codes

Our aim is to improve the rate-distance tradeoff to $\rho = \tilde{\Theta}(\varepsilon)$. Intuitively, the power of the Reed–Solomon list-decoding algorithm comes from the fact that we can interpolate the q points $(y, r(y))$ of the received word using a *bivariate* polynomial Q of degree roughly $\sqrt{q}$ in each variable (think of $d = O(1)$ for now). If we could use m variables instead of 2, then the degrees would only have to be around $q^{1/m}$.

First attempt: Replace Step 1 with finding an $(m+1)$-variate polynomial $Q(Y, Z_1, \ldots, Z_m)$ of degree d_Y in Y and d_Z in each Z_i such that $Q(y, r(y), r(y), \ldots, r(y)) = 0$ for every $y \in \mathbb{F}_q$. Then, we will be able to choose a nonzero polynomial Q of degree roughly $q^{1/m}$ such that $Q(Y, f(Y), \ldots, f(Y)) = 0$ for every $f \in \text{LIST}(r, \varepsilon)$, and hence it follows that $Z - f(Y)$ divides the bivariate polynomial $Q^*(Y, Z) = Q(Y, Z, \ldots, Z)$. Unfortunately, Q^* might be the zero polynomial even if Q is nonzero.

Second attempt: Replace Step 1 with finding an $(m+1)$-variate polynomial $Q(Y, Z_1, \ldots, Z_m)$ of degree d_Y in Y and $d_Z = h - 1$ in each Z_i such that $Q(y, r(y), r(y)^h, r(y)^{h^2}, \ldots, r(y)^{h^{m-1}}) = 0$ for every $y \in \mathbb{F}_q$. Then, it follows that $Q^*(Y, Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}})$ is nonzero if Q is nonzero because every monomial in Q with individual degrees at most $h - 1$ in $Z_1, \cdots, Z_m$ gets mapped to a different power of Z. However, here the difficulty is that the degree of $Q^*(Y, f(Y))$ is too high (roughly $d^* = d_Y + d \cdot h^m > d_Z^m$) for us to satisfy the constraint $\varepsilon q \geq d^*$.

Parvaresh–Vardy codes get the best of both worlds by providing more information with each symbol — not just the evaluation of f at each point, but the evaluation of $m - 1$ other polynomials $f_1, \ldots, f_{m-1}$, each of which is still of degree d (as is good for arguing that

$Q(Y, f(Y), f_1(Y), \ldots, f_{m-1}(Y)) = 0$, but can be viewed as raising f to successive powers of h for the purposes of ensuring that Q^* is nonzero.

To introduce this idea, we need some additional algebra.

- For univariate polynomials $f(Y)$ and $E(Y)$, we define $f(Y) \bmod E(Y)$ to be the remainder when f is divided by E. If $E(Y)$ is of degree k, then $f(Y) \bmod E(Y)$ is of degree at most $k-1$.
- The ring $\mathbb{F}[Y]/E(Y)$ consists of all polynomials of degree at most $k-1$ with arithmetic modulo $E(Y)$ (analogous to $\mathbb{Z}_n$ consisting integers smaller than n with arithmetic modulo n). If E is irreducible then, $\mathbb{F}[Y]/E(Y)$ is a field (analogous to $\mathbb{Z}_p$ being a field when p is prime). Indeed, this is how the finite field of size p^k is constructed: take $\mathbb{F} = \mathbb{Z}_p$ and $E(Y)$ to be an irreducible polynomial of degree k over $\mathbb{Z}_p$, and then $\mathbb{F}[Y]/E(Y)$ is the (unique) field of size p^k.
- A multivariate polynomial $Q(Y, Z_1, \ldots, Z_m)$ can be reduced modulo $E(Y)$ by writing it as a polynomial in variables $Z_1, \ldots, Z_m$ with coefficients in $\mathbb{F}[Y]$ and then reducing each coefficient modulo $E(Y)$. After reducing Q modulo E, we think of Q as a polynomial in variables $Z_1, \ldots, Z_m$ with coefficients in the field $\mathbb{F}[Y]/E(Y)$.

Construction 5.21 (Parvaresh–Vardy Codes). For a prime power q, integers $m, d, h \in \mathbb{N}$, and an irreducible polynomial $E(Y)$ of degree larger than d, the q-ary *Parvaresh–Vardy Code* of degree d, power h, redundancy m, and irreducible E is defined as follows:

- The alphabet is $\Sigma = \mathbb{F}_q^m$.
- The blocklength is $\hat{n} = q$.
- The message space is $\mathbb{F}_q^{d+1}$, where we view each message as representing a polynomial $f(Y)$ of degree at most d over $\mathbb{F}_q$.
- For $y \in \mathbb{F}_q$, the y'th symbol of the $\mathrm{Enc}(f)$ is

$$[f_0(y), f_1(y), \ldots, f_{m-1}(y)],$$

where $f_i(Y) = f(Y)^{h^i} \bmod E(Y)$.

Theorem 5.22. For every prime power q, integer $0 \le d < q$, and irreducible polynomial E of degree $d+1$, the q-ary Parvaresh–Vardy code of degree d, redundancy $m = \lceil \log(q/d) \rceil$, power $h = 2$, and irreducible E has has rate $\rho = \tilde{\Omega}(d/q)$ and can be list-decoded in polynomial time up to distance $\delta = 1 - \tilde{O}(d/q)$.

Proof. We are given a received word $r : \mathbb{F}_q \to \mathbb{F}_q^m$, and want to find all elements of $\mathrm{LIST}(r, \varepsilon)$, for some $\varepsilon = \tilde{O}(d/q)$.

Step 1: Find a low-degree Q "explaining" r. We find a polynomial $Q(Y, Z_0, \ldots, Z_{m-1})$ of degree at most d_Y in its first variable Y and at most $h-1$ in each of the remaining variables, and satisfying $Q(y, r(y)) = 0$ for all $y \in \mathbb{F}_q$.

This is possible provided

$$d_Y \cdot h^m > q. \tag{5.1}$$

Moreover, we may assume that Q is not divisible by $E(Y)$. If it is, we can divide out all the factors of $E(Y)$, which will not affect the conditions $Q(y, r(y)) = 0$ since E has no roots (being irreducible).

Step 2: Argue that each $f(Y) \in \mathrm{LIST}(r, \varepsilon)$ is a "root" of a related univariate polynomial Q^*. First, we argue as before that for $f \in \mathrm{LIST}(r, \varepsilon)$, we have

$$Q(Y, f_0(Y), \ldots, f_{m-1}(Y)) = 0. \tag{5.2}$$

Since each f_i has degree at most $\deg(E) - 1 = d$, this will be ensured provided

$$\varepsilon q \geq d_Y + (h-1) \cdot d \cdot m. \tag{5.3}$$

Once we have this, we can reduce both sides of Equation (5.2) modulo $E(Y)$ and deduce

$$\begin{aligned} 0 &= Q(Y, f_0(Y), f_1(Y), \ldots, f_{m-1}(Y)) \bmod E(Y) \\ &= Q(Y, f(Y), f(Y)^h, \ldots, f(Y)^{h^{m-1}}) \bmod E(Y) \end{aligned}$$

Thus, if we define the univariate polynomial

$$Q^*(Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}}) \bmod E(Y),$$

then $f(Y)$ is a root of Q^* over the field $\mathbb{F}_q[Y]/E(Y)$.

Observe that Q^* is nonzero because Q is not divisible by $E(Y)$ and has degree at most $h-1$ in each Z_i. Thus, we can find all elements of $\mathrm{LIST}(r, \varepsilon)$ by factoring $Q^*(Z)$.

For this algorithm to work, we need to satisfy Conditions (5.1) and (5.3). We can satisfy Condition (5.3) by setting $d_Y = \lceil \varepsilon q - dhm \rceil$, in which case Condition (5.1) is satisfied for

$$\varepsilon = \frac{1}{h^m} + \frac{dhm}{q} = \tilde{O}(d/q), \tag{5.4}$$

for $h = 2$ and $m = \lceil \log(q/d) \rceil$. Observing that the rate is $\rho = d/(mq) = \tilde{\Omega}(d/q)$, this completes the proof of the theorem. □

Note that the obstacles to obtaining an tradeoff $\rho \approx \varepsilon$ are the factor of m in expression for the rate $\rho = d/(mq)$ and the factor of hm in Equation (5.4) for ε. We remedy these in the next section.

5.2.4 Folded Reed–Solomon Codes

In this section, we show how to obtain an optimal rate-distance tradeoff, where the rate ρ is arbitrarily close to the agreement parameter ε.

Consider the Parvaresh–Vardy construction with irreducible polynomial $E(Y) = Y^{q-1} - \gamma$, where γ is a generator of the multiplicative group $\mathbb{F}_q^*$. (That is, $\{\gamma, \gamma^2, \ldots, \gamma^{q-1}\} = \mathbb{F}_q \setminus \{0\}$.) Then it turns out that $f^q(Y) = f(\gamma Y) \bmod E(Y)$. So, we set $h = q$ and the degree of $f_i(Y) = f^{h^i}(Y) \bmod E(Y) = f(\gamma^i Y)$ is d even though E has degree $q-1$. For each nonzero element y of $\mathbb{F}_q$, the y'th symbol of the PV encoding of $f(Y)$ is then

$$[f(y), f(\gamma y), \ldots, f(\gamma^{m-1} y)] = [f(\gamma^j), f(\gamma^{j+1}), \ldots, f(\gamma^{j+m-1})], \tag{5.5}$$

where we write $y = \gamma^j$.

Thus, the symbols of the PV encoding have a lot of overlap. For example, the γ^j'th symbol and the γ^{j+1}'th symbol share all but one component. Intuitively, this means that we should only have to send a $1/m$ fraction of the symbols of the codeword, saving us a factor of m in the rate. (The other symbols can be automatically filled in by the receiver.) Thus, the rate becomes $\rho = d/q$, just like in Reed–Solomon codes.

More formally, we use the following codes.

Construction 5.23 (Folded Reed–Solomon Codes). For a prime power q, a generator γ of $\mathbb{F}_q^*$, integers $m, d \in \mathbb{N}$, the q-ary *folded Reed–Solomon code* of degree d, folding parameter m, and generator γ is defined as follows:

- The alphabet is $\Sigma = \mathbb{F}_q^m$.
- The blocklength is $\hat{n} = \lfloor (q-1)/m \rfloor$.
- The message space is $\mathbb{F}_q^{d+1}$, where we view each message as representing a polynomial $f(Y)$ of degree at most d over $\mathbb{F}_q$.
- For $k \in [\hat{n}]$, the k'th symbol of $\mathrm{Enc}(f)$ is

$$[f(\gamma^{(k-1)m}), f(\gamma^{(k-1)m+1}), \ldots, f(\gamma^{k \cdot m - 1})].$$

We now show that these codes can be efficiently list-decoded at distance arbitrarily close to $1 - \rho$, where $\rho = d/q$ is the rate.

Theorem 5.24. For every prime power q, integers $0 \le d, m \le q$, and generator γ of $\mathbb{F}_q^*$, the q-ary folded Reed–Solomon code of degree d, folding parameter m, and generator γ has rate at least $\rho = d/q$ and can be can be list-decoded in time $q^{O(m)}$ up to distance $\delta = 1 - d/q - O(\sqrt{1/m})$.

Note that this theorem is most interesting (and improves on the Reed–Solomon decoding of Theorem 5.19) when m is larger than q/d, in contrast to the setting of parameters in Theorem 5.22, where m is logarithmic in q. We can afford the larger setting of m because now the rate ρ does not depend on m, and it is this change in parameters that enables us to save us the factor of hm in Equation (5.4). However, the running time (and alphabet size) does grow exponentially with m.

Proof. We are given a received word $r : [\hat{n}] \to \mathbb{F}_q^m$, and want to find all elements of $\mathrm{LIST}(r, \varepsilon)$ for an appropriate choice of $\varepsilon = d/q + O(\sqrt{1/m})$.

Step 0: Unpack to a PV received word. From r, we will obtain a received word $r' : \mathbb{F}_q \to \mathbb{F}_q^{m'}$ for the Parvaresh–Vardy code of degree d, power $h = q$, redundancy $m' = \lfloor \sqrt{m-1} \rfloor$, and irreducible $E(Y) = Y^{q-1} - \gamma$. Specifically, following Equation (5.5), each symbol of r yields $m - m'$ symbols of r'. If r agrees with the FRS encoding of f in more than $\varepsilon \hat{n}$ positions, then r' will agree with the PV encoding of f in more than $\varepsilon \hat{n} \cdot (m - m')$ positions. We have

$$\varepsilon \hat{n} \cdot (m - m') \ge \varepsilon \cdot ((q-1)/m - 1) \cdot (m - m') = (\varepsilon - O(1/\sqrt{m})) \cdot q.$$

Thus our task is now to find $\mathrm{LIST}(r', \varepsilon')$ for $\varepsilon' = \varepsilon - O(1/\sqrt{m})$, which we do in a similar manner to the decoding algorithm for Parvaresh–Vardy codes.

Step 1: Find a low-degree Q "explaining" r'. We find a nonzero polynomial $Q(Y, Z_0, \ldots, Z_{m'-1})$ satisfying $Q(y, r'(y)) = 0$ for all $y \in \mathbb{F}_q$ and such that Q has degree at most d_Y in its first variable Y and *total degree* at most $d_Z = 1$ in the Z_i variables. That is, Q only has monomials of the form $Y^j Z_i$ for $j \leq d_Y$. The number of these monomials is $(d_Y + 1) \cdot m'$, so we can find such a Q provided:

$$(d_Y + 1) \cdot m' > q. \tag{5.6}$$

As before, we may assume that Q is not divisible by $E(Y)$. (Note that using total degree 1 instead of individual degrees $h-1$ in the Z_i's requires an exponentially larger setting of m' compared to the setting of m needed for Equation (5.1). However, this will provide a significant savings below. In general, it is best to consider a combination of the two constraints, requiring that the total degree in the Z_i's is at most some d_Z *and* that the individual degrees are all at most $h - 1$.)

Step 2: Argue that each $f(Y) \in \text{LIST}(r', \varepsilon')$ is a "root" of a related univariate polynomial Q^*. First, we argue as before that for $f \in \text{LIST}(r', \varepsilon')$, we have

$$Q(Y, f_0(Y), \ldots, f_{m-1}(Y)) = 0, \tag{5.7}$$

where $f_i(Y) = f^{h^i}(Y) \bmod E(Y) = f(\gamma^i Y)$. Since each f_i has degree at most d, this will be ensured provided

$$\varepsilon' q \geq d_Y + d. \tag{5.8}$$

Note the savings of the factor $(h-1) \cdot m$ as compared to Inequality (); this is because we chose Q to be of total degree 1 in the Z_i's instead of having individual degree 1.

Now, as in the Parvaresh–Vardy decoding, we can reduce both sides of Equation (5.7) modulo $E(Y)$ and deduce

$$\begin{aligned} 0 &= Q(Y, f_0(Y), f_1(Y), \ldots, f_{m-1}(Y)) \bmod E(Y) \\ &= Q(Y, f(Y), f(Y)^h, \ldots, f(Y)^{h^{m-1}}) \bmod E(Y). \end{aligned}$$

Thus, if we define the univariate polynomial

$$Q^*(Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}}) \bmod E(Y),$$

then $f(Y)$ is a root of Q^* over the field $\mathbb{F}_q[Y]/E(Y)$.

Observe that Q^* is nonzero because Q is not divisible by $E(Y)$ and has degree at most $h-1$ in each Z_i. Thus, we can find all elements of $\text{LIST}(r', \varepsilon')$ by factoring $Q^*(Z)$.

For this algorithm to work, we need to satisfy Conditions (5.6) and (5.8). We can satisfy Condition (5.6) by setting $d_Y = \lfloor q/m' \rfloor$, in which case Condition (5.8) is satisfied for

$$\varepsilon' \geq 1/m' + d/q.$$

Recalling that $\varepsilon' = \varepsilon - O(1/\sqrt{m})$ and $m' = \lfloor \sqrt{m-1} \rfloor$, we can take $\varepsilon = d/q + O(1/\sqrt{m})$. □

Setting parameters appropriately gives:

Theorem 5.25. The following holds for all *constants* $\varepsilon, \gamma > 0$. For every $n \in \mathbb{N}$, there is an explicit code of message length n and rate $\rho = \varepsilon - \gamma$ that can be list-decoded in polynomial time from distance $1 - \varepsilon$, with alphabet size $q = \text{poly}(n)$.

The polynomials in the running time and alphabet size (and list size) depend exponentially on γ; specifically they are of the form $n^{O(1/\gamma^2)}$. An optimized version of the decoding algorithm (using some of the ideas of Problem 5.6) can achieve similar results where the relationship between rate and agreement is multiplicative rather than additive (i.e. $\rho = (1 - \gamma) \cdot \varepsilon$). With a variant of code concatenation (Problem 5.2) based on expanders, it is known how to achieve an alphabet size that is independent of n, namely $q = 2^{\text{poly}(1/\gamma)}$. However, for a fixed constant-sized alphabet, e.g. $q = 2$, it is still not known how to achieve list-decoding capacity.

Open Problem 5.26. For any desired constants $\rho, \delta > 0$ such that $\rho > 1 - H_2(\delta)$, construct an explicit family of of codes $\text{Enc}_n : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ that have rate at least ρ and are δ-list-decodable in polynomial time.

5.3 List-decoding views of samplers and expanders

In this section, we show how averaging samplers and expander graphs can be understood as variants of list-decodable codes. The general list decoding framework we use to describe these connections will also allow us to capture the other pseudorandom objects we study in this survey.

We being with the syntactic correspondence between the three objects, so we can view each as a function $\Gamma : [N] \times [D] \to [M]$:

Construction 5.27 (Syntactic Correspondence of Expanders, Samplers, and Codes).

(1) Given a bipartite multigraph G with N left vertices, M right vertices, and left-degree D, we let $\Gamma : [N] \times [D] \to [M]$ be its neighbor function, i.e. $\Gamma(x, y)$ is the y'th neighbor of x in G.

(2) Given a sampler $\text{Samp} : [N] \to [M]^D$, we define $\Gamma : [N] \times [D] \to [M]$ by

$$\Gamma(x, y) = \text{Samp}(x)_y.$$

(3) Given a code $\text{Enc} : [N] \to [q]^D$, we set $M = q \cdot D$ and define $\Gamma : [N] \times [D] \to [M]$ by

$$\Gamma(x, y) = (y, \text{Enc}(x)_y).$$

The correspondence between expanders and samplers is identical to the one from Problem 4.7, which shows that the vertex expansion of G is equivalent to the hitting sampler properties of Samp.

Note that codes correspond to expanders and samplers where the first component of a neighbor/sample equals the edge-label/sample-number. Conversely, any such expander/sampler can be viewed as a code. Many constructions of expanders and samplers have or can be easily modified to have this property. This syntactic constraint turns out to be quite natural in the setting of samplers.

It corresponds to the case where we have D functions $f_1, \ldots, f_D : [M] \to [0,1]$, we are interested in estimating the total average $(1/D)\sum_i \mu(f_i)$, but can only evaluate each f_i on a single sample.

We now consider the following generalized notion of list decoding.

Definition 5.28. For a function $\Gamma : [N] \to [D] \times [M]$, a set $T \subseteq [M]$, and $\varepsilon \in [0,1)$, we define

$$\begin{aligned} \mathrm{LIST}_\Gamma(T, \varepsilon) &= \{x : \Pr_y[\Gamma(x,y) \in T] > \varepsilon\} \text{ and} \\ \mathrm{LIST}_\Gamma(T, 1) &= \{x : \forall y\ \Gamma(x,y) \in T]\}. \end{aligned}$$

More generally, for a function $f : [M] \to [0,1]$, we define

$$\mathrm{LIST}_\Gamma(f, \varepsilon) = \{x : \mathop{\mathrm{E}}_y[f(\Gamma(x,y))] > \varepsilon\}.$$

We can formulate the list-decoding property of Enc, the averaging sampler property of Samp, and the vertex expansion property of Γ in this language as follows:

Proposition 5.29. Let Enc and Γ be as in Construction 5.27. Then Enc is $(1 - 1/q - \varepsilon, K)$ list-decodable iff for every $r \in [M]^D$, we have

$$|\mathrm{LIST}_\Gamma(T_r, 1/q + \varepsilon)| \leq K,$$

where $T_r = \{(y, r_y) : y \in [D]\}$.

Proof. Observe that, for each $x \in [N]$ and $r \in [M]^D$,

$$\begin{aligned} \Pr_y[\Gamma(x,y) \in T_r] &= \Pr_y[(y, \mathrm{Enc}(x)_y) \in T_r] \\ &= \Pr_y[\mathrm{Enc}(x)_y = r_y] \\ &= \mathrm{agr}(\mathrm{Enc}(x), r). \end{aligned}$$

Thus $|\mathrm{LIST}(T_r, 1/q + \varepsilon)| \leq K$ if and only if there are at most K messages x whose encodings have agreement greater than $1/q + \varepsilon$ with r, which is the same as being $(1 - 1/M - \varepsilon, K)$ list-decodable. □

Proposition 5.30. Let Samp and Γ be as in Construction 5.27. Then

(1) Samp is a (δ, ε) averaging sampler iff for every function $f : [M] \to [0,1]$, we have

$$|\mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)| \leq \delta N.$$

(2) Samp is a (δ, ε) boolean averaging sampler iff for every set $T \subseteq [M]$, we have

$$|\mathrm{LIST}_\Gamma(T, \mu(T) + \varepsilon)| \leq \delta N.$$

Proof. Observe that $x \in \mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)$ if and only if x is a "bad" set of coin tosses for Samp — namely $(1/D) \cdot \sum_{i=1}^D f(z_i) > \mu(f) + \varepsilon$, where $(z_1, \ldots, z_D) = \mathrm{Samp}(x)$. Thus Samp errs with probability at most δ over $x \stackrel{\mathrm{R}}{\leftarrow} U_{[N]}$ if and only if $|\mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)| \leq \delta N$. □

Noting that for the sets T_r in Proposition 5.29 have density $\mu(T_r) = 1/q$, we see that the averaging-sampler property implies the standard list-decoding property:

Corollary 5.31. If Samp is a (δ, ε) boolean averaging sampler of the form $\mathrm{Samp}(x, y) = (y, \mathrm{Enc}(x)_y)$, then Enc is $(1 - 1/q - \varepsilon, \delta N)$ list-decodable.

Note, however, that the typical settings of parameters of samplers and list-decodable codes are very different. With codes, we want the alphabet size q to be as small as possible (e.g. $q = 2$) and the blocklength D to be linear or polynomial in the message length $n = \log N$, so $M = qD$ is also linear or polynomial in $n = \log N$. In contrast, we usually are interested in samplers for functions on exponentially large domains (e.g. $M = 2^{\Omega(n)}$).

In Chapter 6, we will see a converse to Corollary **??** when the alphabet size is small: if Enc is $(1 - 1/q - \varepsilon, \delta N)$, list-decodable, then Samp is an $(\delta/\varepsilon, q \cdot \varepsilon)$ averaging sampler.

For expanders, it will be convenient to state the list-decoding property in terms of the following variant of vertex expansion, where we only require that sets of size exactly K expand:

Definition 5.32. For $K \in \mathbb{N}$, a bipartite multigraph G is an $(= K, A)$ *vertex expander* if all sets S consisting of K left-vertices, the neighborhood $N(S)$ is of size at least $A \cdot K$.

Thus, G is a (K, A) vertex expander in the sense of Definition 4.3 iff G is an $(= K', A)$ vertex expander for all positive integers $K' \leq K$.

Lemma 5.33. For $K \in \mathbb{N}$, $\Gamma : [N] \times [D] \to [M]$ is an $(= K, A)$ vertex expander iff for every set $T \subseteq [D] \times [M]$ such that $|T| < KA$, we have:

$$|\mathrm{LIST}(T, 1)| < K.$$

Proof.

$$\begin{aligned}
\Gamma \text{ not an } (= K, A) \text{ expander} &\Leftrightarrow \exists S \subseteq [N] \text{ s.t. } |S| = K \text{ and } |N(S)| < KA \\
&\Leftrightarrow \exists S \subseteq [N] \text{ s.t. } |S| \geq K \text{ and } |N(S)| < KA \\
&\Leftrightarrow \exists T \subseteq [D] \times [M] \text{ s.t. } |\mathrm{LIST}(T, 1)| \geq K \text{ and } |T| < KA,
\end{aligned}$$

where the last equivalence follows because if $T = N(S)$, then $S \subseteq \mathrm{LIST}(T, 1)$, and conversely if $S = \mathrm{LIST}(T, 1)$ then $N(S) \subseteq T$. □

On one hand, this list-decoding property seems easier to establish than the ones for codes and samplers because we look at $\mathrm{LIST}(T, 1)$ instead of $\mathrm{LIST}(T, \mu(T) + \varepsilon)$. On the other hand, to get expansion (i.e. $A > 1$), we require a very tight relationship between $|T|$ and $|\mathrm{LIST}(T, 1)|$. In the setting of codes or samplers, we would not care much about a factor of 2 loss in $|\mathrm{LIST}(T)|$, as this just corresponds to a factor of 2 in list size or error probability. But here it corresponds to a factor 2 loss in expansion, which can be quite significant. In particular, we cannot afford it if we are trying to get $A = (1 - \varepsilon) \cdot D$, as we will be in the next section.

5.4 Expanders from Parvaresh–Vardy Codes

Despite the substantial differences in the standard settings of parameters between codes, samplers, and expanders, it can be very useful to translates ideas and techniques from one object to the other using the connections described in the previous section. In particular, in this section we will see how to build graphs with extremely good vertex expansion ($A = (1-\varepsilon)D$) from Parvaresh–Vardy codes.

Consider the bipartite multigraph obtained from the Parvaresh–Vardy codes (Construction 5.21) via the correspondence of Construction 5.27. That is, we define a neighbor function $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q \times \mathbb{F}_q^m$ by

$$\Gamma(f, y) = [y, f_0(y), f_1(y), \ldots, f_{m-1}(y)], \tag{5.9}$$

where $f(Y)$ is a polynomial of degree at most $n-1$ over $\mathbb{F}_q$, and we define $f_i(Y) = f(Y)^{h^i} \bmod E(Y)$, where E is a fixed irreducible polynomial of degree n over $\mathbb{F}_q$. (Note that we are using $n-1$ instead of d to denote degree of f.)

Theorem 5.34. Let $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q^{m+1}$ be the neighbor function of the bipartite multigraph corresponding to the q-ary Parvaresh–Vardy code of degree d, power h, and redundancy m via Construction 5.27. Then Γ is a (K_{max}, A) expander for $K_{max} = h^m$ and $A = q - nhm$.

Proof. Let K be any integer less than or equal to $K_{max} = h^m$, and let $A = q - nmh$. By Lemma 5.33, it suffices to show that for every set $T \subseteq \mathbb{F}_q^{m+1}$ of size at most $AK - 1$, we have $|\mathrm{LIST}(T)| \le K - 1$.

We begin by doing the proof for $K = K_{max} = h^m$, and later describe the modifications to handle smaller values of K. The proof goes along the same lines as the list-decoding algorithm for the Parvaresh–Vardy codes from Section 5.2.3.

Step 1: Find a low-degree Q vanishing on T. We find a nonzero polynomial $Q(Y, Z_0, \ldots, Z_{m-1})$ of degree at most $d_Y = A - 1$ in its first variable Y and at most $h-1$ in each of the remaining variables such that $Q(z) = 0$ for all $z \in T$. (Compare this to $Q(y, r(y)) = 0$ for all $y \in \mathbb{F}_q$ in the list-decoding algorithm, which corresponds to taking $T = T_r$.)

This is possible because

$$A \cdot h^m = AK > |T|.$$

Moreover, we may assume that Q is not divisible by $E(Y)$. If it is, we can divide out all the factors of $E(Y)$, which will not affect the conditions $Q(z) = 0$ since E has no roots (being irreducible).

Step 2: Argue that each $f(Y) \in \mathrm{LIST}(r)$ is a "root" of a related univariate polynomial Q^*. First, we argue as in the list-decoding algorithm that if $f \in \mathrm{LIST}(r, 1)$, we have

$$Q(Y, f_0(Y), \ldots, f_{m-1}(Y)) = 0.$$

This is ensured because

$$q > A - 1 + nmh.$$

(In the list-decoding algorithm, the left-hand side of this inequality was εq, since we were bounding $|\mathrm{LIST}(T_r, \varepsilon)|$.)

Once we have this, we can reduce both sides modulo $E(Y)$ and deduce

$$\begin{aligned} 0 &= Q(Y, f_0(Y), f_2(Y), \ldots, f_{m-1}(Y)) \bmod E(Y) \\ &= Q(Y, f(Y), f(Y)^2, \ldots, f(Y)^{m-1}) \bmod E(Y) \end{aligned}$$

Thus, if we define the univariate polynomial

$$Q^*(Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}}) \bmod E(Y),$$

then $f(Y)$ is a root of Q^* over the field $\mathbb{F}_q[Y]/E(Y)$.

Observe that Q^* is nonzero because Q is not divisible by $E(Y)$ and has degree at most $h-1$ in each Z_i. Thus,

$$|\mathrm{LIST}(T, 1)| \leq \deg(Q^*) \leq h - 1 + (h-1) \cdot h + (h-1) \cdot h^2 + \cdots + (h-1) \cdot h^{m-1} = K - 1.$$

(Compare this to the list-decoding algorithm, where our primary goal was to efficiently enumerate the elements of $\mathrm{LIST}(T, \varepsilon)$, as opposed to bound its size.)

Handling smaller values of K. We further restrict $Q(Y, Z_1, \ldots, Z_m)$ to only have nonzero coefficients on monomials of the form $Y^i \mathrm{Mon}_j(Z_1, \ldots, Z_m)$ for $0 \leq i \leq A-1$ and $0 \leq j \leq K-1 \leq h^m - 1$, where $\mathrm{Mon}_j(Z_1, \ldots, Z_m) = Z_1^{j_0} \cdots Z_m^{j_{m-1}}$ and $j = j_0 + j_1 h + \cdots + j_{m-1} h^{m-1}$ is the base-h representation of j. Note that this gives us $AK > |T|$ monomials, so Step 1 is possible. Moreover $M_j(Z, Z^h, Z^{h^2}, \ldots, Z^{h^{m-1}}) = Z^j$, so the degree of Q^* is at most $K-1$, and we get the desired list-size bound in Step 3. □

Setting parameters, we have:

Theorem 5.35. For every constant $\alpha > 0$, every $N \in \mathbb{N}$, $K \leq N$, and $\varepsilon > 0$, there is an explicit $(K, (1-\varepsilon)D)$ expander with N left-vertices, M right-vertices, left-degree $D = O((\log N)(\log K)/\varepsilon)^{1+1/\alpha}$ and $M \leq D^2 \cdot K^{1+\alpha}$. Moreover, D is a power of 2.

Proof. Let $n = \log N$ and $k = \log K_{\max}$. Let $h = \lceil (2nk/\varepsilon)^{1/\alpha} \rceil$ and let q be the power of 2 in the interval $(h^{1+\alpha}/2, h^{1+\alpha}]$.

Set $m = \lceil (\log K_{\max})/(\log h) \rceil$, so that $h^{m-1} \leq K_{\max} \leq h^m$. Then, by Theorem 5.34, the graph $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q^{m+1}$ defined in (5.9) is an (h^m, A) expander for $A = q - nhm$. Since $K_{\max} \leq h^m$, it is also a (K_{max}, A) expander.

Note that the number of left-vertices in Γ is $q^n \geq N$, and the number of right-vertices is

$$M = q^{m+1} \leq q^2 \cdot h^{(1+\alpha)\cdot(m-1)} \leq q^2 \cdot K_{\max}^{1+\alpha}.$$

The degree is

$$D = q \leq h^{1+\alpha} = O(nk/\varepsilon)^{1+1/\alpha} = O((\log N)(\log K_{\max})/\varepsilon)^{1+1/\alpha}.$$

To see that the expansion factor $A = q - nhm \geq q - nhk$ is at least $(1-\varepsilon)D = (1-\varepsilon)q$, note that

$$nhk \leq (\varepsilon/2) \cdot h^{1+\alpha} \leq \varepsilon q,$$

where the first inequality holds because $h^\alpha \geq 2nk/\varepsilon$.

Finally, the construction is explicit because a description of $\mathbb{F}_q$ for q a power of 2 (i.e. an irreducible polynomial of degree $\log q$ over $\mathbb{F}_2$) as well as an irreducible polynomial $E(Y)$ of degree n over $\mathbb{F}_q$ can be found in time $\text{poly}(n, \log q) = \text{poly}(\log N, \log D)$. □

These expanders are of polylogarithmic rather than constant degree. But the expansion is almost as large as possible given the degree ($A = (1-\varepsilon)\cdot D$), and the size of the right-hand side is almost as small as possible (in a (K, A) expander, we must have $M \geq KA = (1-\varepsilon)KD$). In particular, these expanders are quite good for the data structure application of Problem 4.10 — storing a K-sized subset of $[N]$ using $K^{1.01} \cdot \text{polylog}(N)$ bits in such a way that membership can be probabilistically tested by reading 1 bit of the data structure. (An efficient solution to that application actually requires more than the graph being explicit in the usual sense, but also that there are efficient algorithms for finding all left-vertices having at least some δ fraction neighbors in a given set $T \subseteq [M]$ of right vertices, but the expanders above can be shown to have that property by a variant of the list-decoding algorithm above.)

A deficiency of the expander of Theorem 5.35 is that the size of the right-hand side is polynomial in K and D (for constant α), whereas the optimal bound is $M = O(KD/\varepsilon)$. Achieving the latter, while keeping the left-degree polylogarithmic, is an open problem:

Open Problem 5.36. Construct ($= K, A$) bipartite expanders with N left-vertices, degree $D = \text{poly}(\log N)$, expansion $A = .99D$, and $M = O(KD)$ right-hand vertices.

We remark that a construction where D is *quasipolynomial* in $\log N$ is known.

5.5 Exercises

~~**Problem 5.1 (Limits of List Decoding).** Show that if there exists a q-ary code $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ of rate ρ that is (δ, L) list-decodable, then $\rho \leq 1 - H_q(\delta, \hat{n}) + (\log_q L)/\hat{n}$~~

Problem 5.2. (Concatenated Codes) For codes $\text{Enc}_1 : \{1, \ldots, N\} \to \Sigma_1^{n_1}$ and $\text{Enc}_2 : \Sigma_1 \to \Sigma_2^{n_2}$, their *concatenation* $\text{Enc} : \{1, \ldots, N\} \to \Sigma_2^{n_1 n_2}$ is defined by

$$\text{Enc}(m) = \text{Enc}_2(\text{Enc}_1(m)_1)\text{Enc}_2(\text{Enc}_1(m)_2) \cdots \text{Enc}_2(\text{Enc}_1(m)_{n_1}).$$

This is typically used as a tool for reducing alphabet size, e.g. with $\Sigma_2 = \{0, 1\}$.

(1) Prove that if Enc_1 has minimum distance δ_1 and Enc_2 has minimum distance δ_2, then Enc has minimum distance at least $\delta_1\delta_2$.

(2) Prove that if Enc_1 is $(1 - \varepsilon_1, \ell_1)$ list-decodable and Enc_2 is (δ_2, ℓ_2) list-decodable, then Enc is $((1 - \varepsilon_1\ell_2) \cdot \delta_2, \ell_1\ell_2)$ list-decodable.

(3) By concatenating a Reed–Solomon code and a Hadamard code, show that for every $n \in \mathbb{N}$ and $\varepsilon > 0$, there is a (fully) explicit code $\text{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ with blocklength $\hat{n} = O(n^2/\varepsilon^2)$ with minimum distance at least $1/2 - \varepsilon$. Furthermore, show that with blocklength $\hat{n} = \text{poly}(n, 1/\varepsilon)$, we can obtain a code that is $(1/2 - \varepsilon, \text{poly}(1/\varepsilon))$ list-decodable in *polynomial time.* (Hint: the inner code can be decoded by brute force.)

Problem 5.3. (List Decoding implies Unique Decoding for Random Errors)

(1) Suppose that $\mathcal{C} \subseteq \{0,1\}^{\hat{n}}$ is a code with minimum distance at least $1/4$ and rate at most $\alpha\varepsilon^2$ for a constant $\alpha > 0$, and we transmit a codeword $c \in \mathcal{C}$ over a channel in which each bit is flipped with probability $1/2 - 2\varepsilon$. Show that if α is a sufficiently small constant (independent of $\hat{n}$ and ε), then all but exponentially small probability over the errors, c will be the unique codeword at distance at most $1/2 - \varepsilon$ from the received word r.
(2) Using Problem 5.2, deduce that for every $\varepsilon > 0$ and $n \in \mathbb{N}$, there is an explicit code of blocklength $\hat{n} = \text{poly}(n, 1/\varepsilon)$ that can be uniquely decoded from $(1/2 - 2\varepsilon)$ random errors as above in polynomial time.

Problem 5.4. (Linear Codes) For a prime power q, a q-ary code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is called *linear* if $\mathcal{C}$ is a linear subspace of $\mathbb{F}_q^{\hat{n}}$. That is, for every $u, v \in \mathcal{C}$ and $\alpha \in \mathbb{F}_q$, we have $u + v \in \mathcal{C}$ and $\alpha u \in \mathcal{C}$.

(1) Verify that the Hadamard, Reed–Solomon, and Reed–Muller codes are all linear.
(2) Show that if $\mathcal{C}$ is linear, then the minimum distance of $\mathcal{C}$ equals the *minimum weight* of $\mathcal{C}$, which is defined to be $\min_{c \in \mathcal{C} \setminus \{0\}} |\{i : c_i \neq 0\}|/\hat{n}$.
(3) Show that if $\mathcal{C}$ is a subspace of dimension n, then its rate is $n/\hat{n}$ and it has has an encoding function $\text{Enc} : \mathbb{F}_q^n \to \mathbb{F}_q^{\hat{n}}$ that is a linear map. Moreover, the encoding function can be made to have the property that there is a set $S \subseteq [\hat{n}]$ of n coordinates such that $\text{Enc}(m)|_S = m$ for every $m \in \mathbb{F}_q^{\hat{n}}$. (Here $c|_S$ is the projection of c onto the coordinates in S.) Such encodings are called *systematic*, and will be useful when we study locally decodable codes in Chapter 7.
(4) Find explicit systematic encodings for the Hadamard and Reed–Solomon codes.
(5) Show that the nonconstructive bound of Theorem 5.8, Part 1, can be achieved by a linear code. That is, for every prime power q, $\hat{n} \in \mathbb{N}$, and $\delta \in (0, 1 - 1/q)$, there exists a *linear* q-ary code of block length $\hat{n}$, minimum distance at least δ, and rate at least $\rho = 1 - H_q(\delta, \hat{n})$. (Hint: chose a random linear map $\text{Enc} : \mathbb{F}_q^n \to \mathbb{F}_q^{\hat{n}}$ and use Part 2.)

Problem 5.5. (LDPC Codes) Given a bipartite multigraph G with N left-vertices and M right-vertices, we can obtain a linear code $\mathcal{C} \subseteq \{0,1\}^N$ (where we view $\{0,1\}$ as the field of two elements) by:

$$\mathcal{C} = \{c \in \{0,1\}^N : \forall j \in [M] \ \oplus_{i \in \Gamma(j)} c_i = 0\},$$

where $\Gamma(j)$ denotes the set of neighbors of vertex j. When G has small left-degree D (e.g. $D = O(1)$), then $\mathcal{C}$ is called a *low-density parity check (LDPC) code.*

(1) Show that $\mathcal{C}$ has rate at least $1 - M/N$.

(2) Show that if G is a (K, A) expander for $A > D/2$, then $\mathcal{C}$ has minimum distance at least $\delta = K/N$.
(3) Show that if G is a $(K, (1-\varepsilon)D)$ expander for a sufficiently small constant ε, then $\mathcal{C}$ has a polynomial-time δ-decoder for $\delta = (1-3\varepsilon) \cdot K/N$. Assume that G is given as input to the decoder. (Hint: given a received word $r \in \{0,1\}^n$, flip all coordinates of r for which at least $2/3$ of the neighboring parity checks are not satisfied, and argue that the number of errors decreases by a constant factor. It may be useful to use the results of Problem 4.10.)

Problem 5.6. (Improved list-decoding of Reed–Solomon Codes)

(1) Show that there is a polynomial-time algorithm for list-decoding the Reed-Solomon codes of degree d over $\mathbb{F}_q$ up to distance $1-\sqrt{2d/q}$, improving the $1-2\sqrt{d/q}$ bound from lecture. (Hint: do not use fixed upper bounds on the individual degrees of the interpolating polynomial $Q(Y, Z)$, but rather allow as many monomials as possible.)
(2) (*) Improve the list-decoding radius further to $1-\sqrt{d/q}$ by using the following "method of multiplicities". First, require the interpolating polynomial $Q(Y, Z)$ to have a zero of multiplicity s at each point $(y, r(y))$ — that is, the polynomial $Q(Y+y, Z+r(y))$ should have no monomials of degree smaller than s. Second, use the fact that a univariate polynomial $R(Y)$ of degree t can have at most t roots, counting multiplicities.

Problem 5.7. (Twenty Questions) In the game of 20 questions, an oracle has an arbitrary secret $s \in \{0,1\}^n$ and the aim is to determine the secret by asking the oracle as few yes/no questions about s as possible. It is easy to see that n questions are necessary and sufficient. Here we consider a variant where the oracle has two secrets $s_1, s_2 \in \{0,1\}^n$, and can adversarially decide to answer each question according to either s_1 or s_2. That is, for a question $f : \{0,1\}^n \to \{0,1\}$, the oracle may answer with either $f(s_1)$ or $f(s_2)$. Here it turns out to be impossible to pin down either of the secrets with certainty, no matter how many questions we ask, but we can hope to compute a small list L of secrets such that $|L \cap \{s_1, s_2\}| \neq \emptyset$. (In fact, $|L|$ can be made as small as 2.) This variant of twenty questions apparently was motivated by problems in Internet traffic routing.

(1) Let $\mathrm{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ be a code such that that every two codewords in Enc *agree* in at least a $1/2-\varepsilon$ fraction of positions and that Enc has a polynomial-time $(1/4+\varepsilon, \ell)$ list-decoding algorithm. Show how to solve the above problem in polynomial time by asking the $\hat{n}$ questions $\{f_i\}$ defined by $f_i(x) = \mathrm{Enc}(x)_i$.
(2) Taking Enc to be the code constructed in Problem 1, deduce that $\hat{n} = \mathrm{poly}(n)$ questions suffices.

5.6 Chapter Notes and References

Standard texts on coding theory include Macwilliams–Sloane [MS2, MS3] and the Handbook of Coding Theory [PHB]. The lecture notes of Sudan [Sud2] present coding theory with an algorithmic perspective, and Guruswami [Gur2] gives a thorough treatment of list decoding.

The field of coding theory began with Shannon's seminal 1948 paper [Sha2], which proposed the study of stochastic error models and proved that a random error-correcting code achieves an optimal rate-distance tradeoff with high probability. The study of decoding from worst-case errors began a couple of years later with the work of Hamming [Ham]. The notion of list decoding was proposed independently in the late 1950's, in the work of Elias [Eli1] and Wozencraft [Woz].

The nonconstructive bound for the tradeoff between rate and minimum distance of Theorem 5.8 (Part 1) is due to Gilbert [Gil1], and its extension to linear codes in Problem 5.4 (Part 5) is due to Varshamov [Var]; together they are known as the Gilbert–Varshamov Bound. The nonconstructive bound for list-decoding of Theorem 5.8 (Part 2) is due to Elias [Eli3], and it has been extended to linear codes in [ZP, GHSZ, GHK]. The Johnson Bound (Proposition 5.10) was proven for binary codes by Johnson [Joh], and was extended to the q-ary case in [GRS]. An optimized form of the bound can be found in [GS3].

The binary ($q = 2$) case of Reed–Muller codes was introduced independently by Reed [Ree] and Muller [Mul] in the mid-50's. Reed–Solomon codes were introduced in 1960 by Reed and Solomon [RS]. Polynomial time unique-decoding algorithms for Reed–Solomon Codes include those of Peterson [Pet] and Berlekamp [Ber2], . The first nontrivial list decoding algorithm was given by Goldreich and Levin [GL]; while their algorithm is stated in the language of "hardcore bits for one-way functions," it can be viewed as an efficient "local" list-decoding algorithm for the Hadamard Code. (Local decoding algorithms are discussed in Chapter 7.) The list-decoding algorithm for Reed–Solomon Codes of Theorem 5.19 is from the seminal work of Sudan [Sud1], which sparked a resurgence in the study of list decoding. The improved decoding algorithm of Problem 5.6, Part 2 is due to Guruswami and Sudan [GS1]. Parvaresh–Vardy codes and their decoding algorithm are from [PV]. Folded Reed–Solomon Codes and their capacity-achieving list-decoding algorithm are due to Guruswami and Rudra [GR]. (In all cases, our presentation uses a simplified setting of parameters compared to the original algorithms.)

The list-decoding views of expanders and samplers emerged out of work on randomness extractors (the subject of Chapter 6). Specifically, a close connection between extractors and expanders was understood already at the time that extractors were introduced by Nisan and Zuckerman [NZ]. An equivalence between extractors and averaging samplers was established by Zuckerman [Zuc2] (building on previous connections between other types of samplers and expanders [Sip2, CW1]). A connection between list-decodable codes and extractors emerged in the work of Trevisan [Tre], and the list-decoding view of extractors was crystallized by Ta-Shma and Zuckerman [TZ]. The list-decoding formulation of vertex expansion and the construction of expanders from Parvaresh–Vardy codes is due to Guruswami, Umans, and Vadhan [GUV].

Code concatenation (Problem 5.2) was introduced by Forney [For], who used it to construct explicit, efficiently decodable binary codes that achieve capacity for random errors. Efficient list-decoding algorithms for the concatenation of a Reed–Solomon code with a Hadamard code were given in [GS2]. Low-density parity check (LDPC) codes (Problem 5.5) were introduced by Gallager [Gal]. The use of expansion to analyze such codes and their decoding algorithm comes from

Sipser and Spielman [SS1]. The surveys of Guruswami [Gur1, Gur2] describe the state of the art in expander-based constructions of codes and in LDPC codes, respectively.

The question of Problem 5.7 (Twenty Questions) was posed in [CGL], motivated by Internet traffic routing applications. The solution using list decoding is from [AGKS].

6

Randomness Extractors

Randomness extractors are functions that extract almost-uniform bits from sources of biased and correlated bits. The original motivation for extractors was to simulate randomized algorithms with weak random sources as might arise in nature. This motivation is still compelling, but extractors have taken on a much wider significance in the years since they were introduced. They have found numerous applications in theoretical computer science beyond this initial motivating one, in areas from cryptography to distributed algorithms to metric embeddings. More importantly from the perspective of this survey, they have played a major unifying role in the theory of pseudorandomness. Indeed, the links between the various pseudorandom objects we are studying in this survey (expander graphs, randomness extractors, list-decodable codes, pseudorandom generators, samplers) were all discovered through work on extractors (even though now we find it more natural to present these links in the language of list decoding, as introduced in Section 5.3).

6.1 Motivation and Definition

6.1.1 Deterministic Extractors

Typically, when we design randomized algorithms or protocols, we assume that all algorithms/parties have access to sources of *perfect* randomness, i.e. bits that are unbiased and completely independent. However, when we implement these algorithms, the physical sources of randomness to which we have access may contain biases and correlations. For example, we may use low-order bits of the system clock, the user's mouse movements, or a noisy diode based on quantum effects. While these sources may have some randomness in them, the assumption that the source is perfect is a strong one, and thus it is of interest to try and relax it.

Ideally, what we would like is a compiler that takes any algorithm A that works correctly when fed perfectly random bits U_m, and produces a new algorithm A' that will work even if it is fed random bits $X \in \{0,1\}^n$ that come from a "weak" random source. For example, if A is a **BPP** algorithm, then we would like A' to also run in probabilistic polynomial time. One way to design such compilers is to design a *randomness extractor* $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that $\mathrm{Ext}(X)$ is

distributed uniformly in $\{0,1\}^m$.

IID-Bit Sources (aka Von Neumann Sources). A simple version of this question was already considered by von Neumann. He looked at sources that consist of boolean random variables $X_1, X_2, \ldots, X_n \in \{0,1\}$ that are independent but biased. That is, for every i, $\Pr[X_i = 1] = \delta$ for some unknown δ. How can such a source be converted into a source of independent, unbiased bits? Von Neumann proposed the following extractor: Break all the variables in pairs and for each pair output 0 if the outcome was 01, 1 if the outcome was 10 , and skip the pair if the outcome was 00 or 11. This will yield an unbiased random bit after $1/\delta$ pairs on average.

Independent-Bit Sources. Lets now look at a bit more interesting class of sources in which all the variables are still independent but the bias is no longer the same. Specifically, for every i, $\Pr[X_i = 1] = \delta_i$ and $0 < \delta \leq \delta_i \leq 1 - \delta$ for some constant $\delta > 0$. How can we deal with such a source?

It can be shown that when we take a parity of ℓ bits from such an independent-bit source, the result approaches an unbiased coin flip exponentially fast in ℓ, i.e. $|\Pr\left[\oplus_{i=1}^{\ell} X_i = 1\right] - 1/2| = 2^{-\Omega(\ell}$. The result is not a perfect coin flip but is as good as one for almost all purposes.

Let's be more precise about the problems we are studying. A *source* on $\{0,1\}^n$ is simply a random variable X taking values in $\{0,1\}^n$. In each of the above examples, there is an implicit *class* of sources being studied. For example, $\text{IndBits}_{n,\delta}$ is the class of sources X on $\{0,1\}^n$ where the bits X_i are independent and satisfy $\delta \leq \Pr[X_i = 1] \leq 1 - \delta$. We could define $\text{IIDBits}_{n,\delta}$ to be the same with the further restriction that all of the X_i's are identically distributed, i.e. $\Pr[X_i = 1] = \Pr[X_j = 1]$ for all i, j, thereby capturing von Neumann sources.

Definition 6.1 (deterministic extractors). [1] Let $\mathcal{C}$ be a class of sources on $\{0,1\}^n$. An ε-*extractor* for $\mathcal{C}$ is a function $\text{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that for every $X \in \mathcal{C}$, $\text{Ext}(X)$ is "ε-close" to U_m.

Note that we want a *single* function Ext that works for all sources in the class. This captures the idea that we do not want to assume we know the exact distribution of the physical source we are using, but only that it comes from some class. For example, for $\text{IndBits}_{n,\delta}$, we know that the bits are independent and none are too biased, but not the specific bias of each bit. Note also that we only allow the extractor *one* sample from the source X. If we want to allow multiple independent samples, then this should be modelled explicitly in our class of sources; ideally we would like to minimize the independence assumptions used.

We still need to define what we mean for the output to be ε-close to U_m.

Definition 6.2. For random variables X and Y taking values in $\mathcal{U}$, their *statistical difference* (also known as *variation distance*) is $\Delta(X,Y) = \max_{T \subseteq \mathcal{U}} |\Pr[X \in T] - \Pr[Y \in T]|$. We say that X and Y are ε-*close* if $\Delta(X,Y) \leq \varepsilon$.

---

[1] Such extractors are called *deterministic* or *seedless* to contrast with the probabilistic or *seeded* randomness extractors we will see later.

Intuitively, any event in X happens in Y with the same probability $\pm\varepsilon$. This is perhaps the most natural measure of distance for probability distributions (much more so than the ℓ_2 distance we used in the study of random walks). In particular, it satisfies the following natural properties.

Lemma 6.3 (properties of statistical difference). Let $X, Y, Z, X_1, X_2, Y_1, Y_2$ be random variables taking values in a universe $\mathcal{U}$. Then,

(1) $\Delta(X,Y) \geq 0$, with equality iff X and Y are identically distributed,
(2) $\Delta(X,Y) \leq 1$, with equality iff X and Y have disjoint supports,
(3) $\Delta(X,Y) = \Delta(Y,X)$,
(4) $\Delta(X,Z) \leq \Delta(X,Y) + \Delta(Y,Z)$,
(5) for every function f, we have $\Delta(f(X), f(Y)) \leq \Delta(X,Y)$,
(6) $\Delta((X_1,X_2),(Y_1,Y_2)) \leq \Delta(X_1,Y_1) + \Delta(X_2,Y_2)$ if X_1 and X_2, as well as Y_1 and Y_2, are independent, and
(7) $\Delta(X,Y) = \frac{1}{2} \cdot |X - Y|_1$, where $|\cdot|_1$ is the ℓ_1 distance. (Thus, X is ε-close to Y iff we can transform X into Y by "shifting" at most an ε fraction of probability mass.)

We now observe that extractors according to this definition give us the "compilers" we want.

Proposition 6.4. Let $A(w;r)$ be a randomized algorithm such that $A(w;U_m)$ has error probability at most γ, and let $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ be an ε-extractor for a class $\mathcal{C}$ of sources on $\{0,1\}^n$. Define $A'(w;x) = A(w;\mathrm{Ext}(x))$. Then for every source $X \in \mathcal{C}$, $A'(w;X)$ has error probability at most $\gamma + \varepsilon$.

This application identifies some additional properties we'd like from our extractors. We'd like the extractor itself to be efficiently computable (e.g. polynomial time). In particular, to get m almost-uniform bits out, we should need at most $n = \mathrm{poly}(m)$ bits bits from the weak random source.

We can cast our earlier extractor for sources of independent bits in this language:

Proposition 6.5. For every constant $\delta > 0$, every $n, m \in \mathbb{N}$, there is a polynomial-time computable function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ that is an ε-extractor for $\mathrm{IndBits}_{n,\delta}$, with $\varepsilon = m \cdot 2^{-\Omega(n/m)}$.

In particular, taking $n = m^2$, we get exponentially small error with a source of polynomial length.

Proof. Ext breaks the source into m blocks of length $\lfloor n/m \rfloor$ and outputs the parity of each block. □

Unpredictable-Bit Sources (aka Santha–Vazirani Sources). Another interesting class of sources, which looks similar to the previous example is the class $\mathrm{UnpredBits}_{n,\delta}$ of *unpredictable-bit sources*. These are the sources that for every i, every $x_1, \ldots, x_n \in \{0,1\}$ and some constant $\delta > 0$, satisfy

$$\delta \leq \Pr\left[X_i = 1 \mid X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}\right] \leq 1 - \delta$$

The parity extractor used above will be of no help with this source since the next bit could be chosen in a way that the parity will be equal to 1 with probability δ. Indeed, there does not exist any nontrivial extractor for these sources — the best we can do is output the first bit:

Proposition 6.6. For every $n \in \mathbb{N}$, $\delta > 0$, and fixed extraction function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}$ there exists a source $X \in \mathrm{UnpredBits}_{n,\delta}$ such that either $\Pr[\mathrm{Ext}(X) = 1] \leq \delta$ or $\Pr[\mathrm{Ext}(X) = 1] \geq 1 - \delta$. That is, there is no ε-extractor for $\mathrm{UnpredBits}_{n,\delta}$ for $\varepsilon < 1/2 - \delta$.

The proof is left as an exercise (Problem 6.6).

Nevertheless, as we will see, the answer to the question whether we can simulate **BPP** algorithms with unpredictable sources will be "yes"! Indeed, we will even be able to handle a much more general class of sources, introduced in the next section.

6.1.2 Entropy Measures and General Weak Sources

Intuitively, to extract m almost-uniform bits from a source, the source must have at least "m bits of randomness" in it (e.g. its support cannot be much smaller than 2^m). Ideally, this is all we would like to assume about a source. Thus, we need some measure of how much randomness is in a random variable; this can be done using various notions of *entropy* described below.

Definition 6.7 (entropy measures). Let X be a random variable. Then

- the *Shannon entropy* of X is:

$$\mathrm{H}_{Sh}(X) = \mathop{\mathrm{E}}_{x \xleftarrow{\mathrm{R}} X}\left[\log \frac{1}{\Pr[X = x]}\right].$$

- the *Rényi entropy* of X is:

$$\mathrm{H}_2(X) = \log\left(\frac{1}{\mathrm{E}_{x \xleftarrow{\mathrm{R}} X}[\Pr[X = x]]}\right) = \log \frac{1}{\mathrm{CP}(X)}, \text{ and}$$

- the *min-entropy* of X is:

$$\mathrm{H}_\infty(X) = \min_x \left\{\log \frac{1}{\Pr[X = x]}\right\},$$

where all logs are base 2.

Rényi entropy $\mathrm{H}_2(X)$ should not be confused with the the binary entropy function $H_2(\delta)$ from Definition 5.6. Indeed, the q-ary entropy $H_q(\delta)$ is equal to the *Shannon* entropy of a random variable that equals 1 with probability $1 - \delta$ and is uniformly distributed in $\{2, \ldots, q\}$ with probability δ.

All the three measures satisfy the following properties we would expect from a measure of randomness:

Lemma 6.8 (properties of entropy). For each of of the entropy measures $\mathrm{H} \in \{\mathrm{H}_{Sh}, \mathrm{H}_2, \mathrm{H}_\infty\}$ and random variables X, Y, we have:

- $\mathrm{H}(X) \geq 0$, with equality iff X is supported on a single element,
- $\mathrm{H}(X) \leq \log |\mathrm{Supp}(X)|$, with equality iff X is uniform on $\mathrm{Supp}(X)$,
- if X, Y are independent, then $\mathrm{H}((X,Y)) = \mathrm{H}(X) + \mathrm{H}(Y)$,
- for every deterministic function f, we have $\mathrm{H}(f(X)) \leq \mathrm{H}(X)$, and
- for every X, we have $\mathrm{H}_\infty(X) \leq \mathrm{H}_2(X) \leq \mathrm{H}_{Sh}(X)$.

To illustrate the differences between the three notions, consider a source X such that $X = 0^n$ with probability 0.99 and $X = U_n$ with probability 0.01. Then $\mathrm{H}_{Sh}(X) \geq 0.01n$ (contribution from the uniform distribution), $\mathrm{H}_2(X) \leq \log(1/.99^2) < 1$ and $\mathrm{H}_\infty(X) \leq \log(1/.99) < 1$ (contribution from 0^n). Note that even though X has Shannon entropy linear in n, we cannot expect to extract bits that are close to uniform or carry out any useful randomized computations with one sample from X, because it gives us nothing useful 99% of the time. Thus, we should use the stronger measures of entropy given by H_2 or H_∞.

Then why is Shannon entropy so widely used in information theory results? The reason is that such results typically study what happens when you have many independent samples from the source (whereas we only allow one sample). In the case of many samples, it turns out that the the source is "close" to one where the min-entropy is roughly equal to the Shannon entropy. Thus the distinction between these entropy measures becomes less significant. (Recall that we only allow one sample from the source.) Moreover, Shannon entropy satisfies many nice identities that make it quite easy to work with. Min-entropy and Rényi entropy are much more delicate.

We will consider the task of extracting randomness from sources where all we know is a lower bound on the min-entropy:

Definition 6.9. A random variable X is a *k-source* is $\mathrm{H}_\infty(X) \geq k$, i.e., if $\Pr[X = x] \leq 2^{-k}$.

A typical setting of parameters is $k = \delta n$ for some fixed δ, e.g., 0.01. We call δ the *min-entropy rate*. Some different ranges that are commonly studied (and are useful for different applications): $k = \mathrm{polylog}(n)$, $k = n^\gamma$ for a constant $\gamma \in (0,1)$, $k = \delta n$ for a constant $\delta \in (0,1)$, and $k = n - O(1)$. The middle two ($k = n^\gamma$ and $k = \delta n$) are the most natural for simulating randomized algorithms with weak random sources.

Examples of k-sources:

- k random and independent bits, together with $n - k$ fixed bits (in an arbitrary order). These are called *oblivious bit-fixing* sources.
- k random and independent bits, and $n - k$ bits that depend arbitrarily on the first k bits. These are called *adaptive bit-fixing sources*.
- Unpredictable-bit sources with bias parameter δ. These are k-sources with $k = \log(1/(1-\delta)^n) = \Theta(\delta n)$.
- Uniform distribution on a set $S \subset \{0,1\}^n$ with $|S| = 2^k$. These are called *flat k-sources*.

It turns out that flat k-sources are really representative of general k-sources.

Lemma 6.10. Every k-source is a convex combination of flat k-sources (provided that $2^k \in \mathbb{N}$), i.e., $X = \sum p_i X_i$ with $0 \leq p_i \leq 1$, $\sum p_i = 1$ and all the X_i are flat k-sources.

Proof Sketch. Let X be a k-source that takes values in a domain of size N (e.g. if X takes values in $\{0,1\}^n$, then $N = 2^n$). View X as an N-dimensional vector, where $X(i)$ is the probability mass of i. Then X is a k-source if and only if $X(i) \in [0, 2^{-k}]$ for every $i \in [N]$ and $\sum_i X(i) = 1$. The set of vectors X satisfying these linear inequalities is a convex polytope. By basic linear programming theory, all of the points in the polytope are convex combinations of its *vertices*, which are defined to be the points that make a maximal subset of the inequalities tight. By inspection, the vertices of the polytope of k-sources are those sources where $X(i) = 2^{-k}$ for 2^k values of i and $X(i) = 0$ for the remaining values of i; these are simply the k-sources. □

By Lemma 6.10, we can think of any k-source as being obtained by first selecting a flat k-source X_i according to some distribution (given by the p_i's) and then selecting a random sample from X_i. This means that if we can compile probabilistic algorithms to work with flat k-sources, then we can compile them to work with any k-source.

6.1.3 Seeded Extractors

Proposition 6.6 tell us that it impossible to have deterministic extractors for unpredictable sources. Here we consider k-sources, which are more general than unpredictable sources, and hence it is also impossible to have deterministic extractors for them. The impossibility result for k-sources is stronger and simpler to prove.

Proposition 6.11. For any $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}$ there exists an $(n-1)$-source X so that $\mathrm{Ext}(X)$ is constant.

Proof. There exists $b \in \{0,1\}$ so that $|\mathrm{Ext}^{-1}(b)| \geq 2^n/2 = 2^{n-1}$. Then let X be the uniform distribution on $\mathrm{Ext}^{-1}(b)$. □

On the other hand, if we reverse the order of quantifiers, allowing the extractor to depend on the source, it is easy to see that good extractors exist and in fact a randomly chosen function will be a good extractor with high probability.

Proposition 6.12. For every $n, k, m \in \mathbb{N}$, every $\varepsilon > 0$, and every flat k-source X, if we choose a random function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ with $m = k - 2\log(1/\varepsilon) - O(1)$, then $\mathrm{Ext}(X)$ will be ε-close to U_m with probability $1 - 2^{-\Omega(K\varepsilon^2)}$, where $K = 2^k$.

(In this chapter, we will make extensive use of the convention that capital variables are 2 raised to the power of the corresponding lowercase variable, such as $K = 2^k$ above.)

Proof. Choose our extractor randomly. We want it to have following property: for all $T \subseteq [M]$, $|\Pr[\mathrm{Ext}(X) \in T] - \Pr[U_m \in T| \leq \varepsilon$. Equivalently, $|\{x \in \mathrm{Supp}(X) : \mathrm{Ext}(x) \in T\}|/K$ differs from the density $\mu(T)$ by at most ε. For each point $x \in \mathrm{Supp}(X)$, the probability that $\mathrm{Ext}(x) \in T$ is

$\mu(T)$, and these events are independent. By the Chernoff Bound (Theorem 2.21) for each fixed T, this condition holds with probability at least $1 - 2^{-\Omega(K\varepsilon^2)}$. Then the probability that condition is violated for at least one T is at most $2^M 2^{-\Omega(K\varepsilon^2)}$, which is less than 1 for $m = k - 2\log(1/\varepsilon) - O(1)$. □

Note that the failure probability is doubly-exponentially small in k. Naively, one might hope that we could get an extractor that's good for all flat k-sources by a union bound. But the number of flat k-sources is $\binom{N}{K} \approx N^K$ (where $N = 2^n$), which is unfortunately a larger double-exponential in k. We can overcome this gap by allowing the extractor to be "slightly" probabilistic, i.e. allowing the extractor a *seed* consisting of a small number of truly random bits in addition to the weak random source. We can think of this seed of truly random bits as a random choice of an extractor from family of extractors. This leads to the following crucial definition:

Definition 6.13 (seeded extractors). Extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-*extractor* if for every k-source X on $\{0,1\}^n$, $\text{Ext}(X, U_d)$ is ε-close to U_m.

(Sometimes we will refer to extractors $\text{Ext} : [N] \times [D] \to [M]$ whose domain and range do not consist of bit-strings. These are defined in the natural way, requiring that $\text{Ext}(X, U_{[D]})$ is ε-close to $U_{[M]}$.)

The goal is to construct extractors that minimize d and maximize m. We prove the following theorem.

Theorem 6.14. For every $n \in \mathbb{N}$, $k \in [0,n]$ and $\varepsilon > 0$, there exists a (k,ε)-extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k + d - 2\log(1/\varepsilon) - O(1)$ and $d = \log(n-k) + 2\log(1/\varepsilon) + O(1)$.

One setting of parameters to keep in mind (for our application of simulating randomized algorithms with a weak source) is $k = \delta n$, with δ a fixed constant (e.g. $\delta = 0.01$), and ε a fixed constant (e.g. $\varepsilon = 0.01$).

Proof. We use the Probabilistic Method. By Lemma 6.10, it suffices for Ext to work for flat k-sources. Choose the extractor Ext at random. Then the probability that the extractor fails is at most the number of flat k-sources times times the probability Ext fails for a fixed flat k-source. By the above proposition, the probability of failure for a fixed flat k-source is at most $2^{-\Omega(KD\varepsilon^2)}$, since (X, U_d) is a flat $(k+d)$-source) and $m = k + d - 2\log(\frac{1}{\varepsilon}) - O(1)$. Thus the total failure probability is at most

$$\binom{N}{K} \cdot 2^{-\Omega(KD\varepsilon^2)} \leq \left(\frac{Ne}{K}\right)^K 2^{-\Omega(KD\varepsilon^2)}.$$

The latter expression is less than 1 if $D\varepsilon^2 \geq c\log(Ne/K) = c \cdot (n-k) + c'$ for constants c, c'.This is equivalent to $d \geq \log(n-k) + 2\log(\frac{1}{\varepsilon}) + O(1)$. □

It turns out that both bounds (on m and d) are individually tight up to the $O(1)$ terms.

Recall that our motivation for extractors was to simulate randomized algorithms given *only* a weak random source, so allowing a truly random seed may seem to defeat the purpose. However, if the seed is of logarithmic length as in Theorem 6.14, then instead of selecting it randomly, we can enumerate all possibilities for the seed and take a majority vote.

Proposition 6.15. Let $A(w;r)$ be a randomized algorithm for computing a function f such that $A(w;U_m)$ has error probability at most γ (i.e. $\Pr[A(w;U_m) \neq f(w)] \leq \gamma]$), and let $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ε)-extractor. Define

$$A'(w;x) = \operatorname*{maj}_{y \in \{0,1\}^d} \{A(w;\mathrm{Ext}(x,y))\}.$$

Then for every k-source X on $\{0,1\}^n$, $A'(w;X)$ has error probability at most $2 \cdot (\gamma + \varepsilon)$.

Proof. The probability that $A(w;\mathrm{Ext}(X,U_d))$ is incorrect is not more than probability $A(w;U_m)$ is incorrect plus ε, i.e. $\gamma + \varepsilon$, by the definition of statistical difference. Then the probability that $\mathrm{maj}_y\, A(w,\mathrm{Ext}(X,y))$ is incorrect is at most $2 \cdot (\gamma + \varepsilon)$, because each error of $\mathrm{maj}_y\, A(w;\mathrm{Ext}(x,y))$ corresponds to $A(w;\mathrm{Ext}(x,U_d))$ erring with probability at least $1/2$. □

Note that the enumeration incurs a 2^d factor slowdown in the simulation. Thus, to retain running time $\mathrm{poly}(m)$, we want to construct extractors where (a) $d = O(\log n)$; (b) Ext is computable in polynomial time; and (c) $m = n^{\Omega(1)}$.

We remark that the error probability in Proposition 6.15 can actually be made exponentially small by using an extractor that is designed for slightly lower min-entropy roughly $k - t$ instead of k. (See Problem 6.2.)

We note that even though seeded extractors suffice for simulating randomized algorithms with only a weak source, they do not suffice for all applications of randomness in theoretical computer science. The trick of eliminating the random seed by enumeration does not work, for example, in cryptographic applications of randomness. Thus the study of deterministic extractors for restricted classes of sources remains a very interesting and active research direction. We, however, will focus on seeded extractors, due to their many applications and their connections to the other pseudorandom objects we are studying.

6.2 Connections to Other Pseudorandom Objects

As mentioned earlier, extractors have played a unifying role in the theory of pseudorandomness, through their close connections with a variety of other pseudorandom objects. In this section, we will see two of these connections. Specifically, how by reinterpreting them appropriately, extractors can be viewed as providing families of hash functions, and as being a certain type of highly expanding graphs.

6.2.1 Extractors as Hash Functions

Proposition 6.12 says that for any subset $S \subseteq [N]$ of size K, if we choose a completely random hash function $h : [N] \to [M]$ for $M \ll K$, then h will map the elements of S almost-uniformly to $[M]$. Equivalently, if we let H be distributed uniformly over all functions $h : [N] \to [M]$ and X be uniform on the set S, then $(H, H(X))$ is statistically close to $(H, U_{[M]})$, where we use the notation U_T to denote the uniform distribution on a set T. Can we use a smaller family of hash functions than the set of all functions $h : [N] \to [M]$? This gives rise to the following variant of extractors.

Definition 6.16 (strong extractors). Extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a *strong* (k,ε)-*extractor* if for every k-source X on $\{0,1\}^n$, $(U_d, \text{Ext}(X, U_d))$ is ε-close to (U_d, U_m). Equivalently, $\text{Ext}'(x,y) = (y, \text{Ext}(x,y))$ is a standard (k,ε)-extractor.

The nonconstructive existence proof of Theorem 6.14 can be extended to establish the existence of very good strong extractors:

Theorem 6.17. For every $n, k \in \mathbb{N}$ and $\varepsilon > 0$ there exists a strong (k,ε)-extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k - 2\log(1/\varepsilon) - O(1)$ and $d = \log(n-k) + 2\log(1/\varepsilon) + O(1)$.

Note that the output length is $m \approx k$ instead of $m \approx k + d$; intuitively a strong extractor needs to extract randomness that is *independent* of the seed and thus can only get the k bits from the source.

We see that strong extractors can be viewed as very small families of hash functions having the almost-uniform mapping property mentioned above. Indeed, our first explicit construction of extractors is obtained by using pairwise independent hash functions.

Theorem 6.18 (Leftover Hash Lemma). If $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is a pairwise independent (or even 2-universal) family of hash functions where $m = k - 2\log(1/\varepsilon)$, then $\text{Ext}(x,h) \stackrel{\text{def}}{=} h(x)$ is a strong (k,ε)-extractor. Equivalently, $\text{Ext}(x,h) = (h, h(x))$ is a standard (k,ε)-extractor.

Note that the seed length equals the number of random bits required to choose $h \stackrel{\text{R}}{\leftarrow} \mathcal{H}$, which is at least n by Problem 3.5.[2] This is far from optimal; for the purposes of simulating randomized algorithms we would like $d = O(\log n)$. However, the output length of the extractor is $m = k - 2\log(1/\varepsilon)$, which is optimal up to an additive constant.

Proof. Let X be an arbitrary k-source on $\{0,1\}^n$, $\mathcal{H}$ as above, and $H \stackrel{\text{R}}{\leftarrow} \mathcal{H}$. Let d be the seed length. We show that $(H, H(X))$ is ε-close to $U_d \times U_m$ in the following three steps:

(1) We show that the collision probability of $(H, H(X))$ is close to that of $U_d \times U_m$.
(2) We note that this is equivalent to saying that the ℓ_2 distance between $(H, H(X))$ and $U_d \times U_m$ is small.
(3) Then we deduce that the statistical difference is small, by recalling that the statistical difference equals half of the ℓ_1 distance, which can be (loosely) bounded by the ℓ_2 distance.

Proof of (1): By definition, $\text{CP}(H, H(X)) = \Pr[(H, H(X)) = (H', H'(X'))]$, where (H', X') is independent of and identically distributed to (H, X). Note that $(H, H(X)) = (H', H'(X))$ if and only if $H = H'$ and either $X = X'$ or $X \neq X'$ but $H(X) = H(X')$. Thus

$$\begin{aligned} \text{CP}(H, H(X)) &= \text{CP}(H) \cdot \Big(\text{CP}(X) + \Pr\left[H(X) = H(X') \mid X \neq X'\right]\Big) \\ &\leq \frac{1}{D} \cdot \left(\frac{1}{K} + \frac{1}{M}\right) \leq \frac{1+\varepsilon^2}{DM}. \end{aligned}$$

[2] Problem 3.5 refers to pairwise independent families, but a similar argument shows that universal families require $\Omega(n)$ random bits. (Instead of constructing orthogonal vectors, we construct vectors that have nonpositive dot product.)

To see the penultimate inequality, note that $\mathrm{CP}(H) = 1/D$ because there are D hash functions, $\mathrm{CP}(X) \le 1/K$ because $\mathrm{H}_\infty(X) \ge k$, and $\Pr[H(X) = H(X') \mid X \ne X'] \le 1/M$ by 2-universality.

Proof of (2):

$$\begin{aligned} \|(H, H(X)) - U_d \times U_m\|^2 &= \mathrm{CP}(H, H(X)) - \frac{1}{DM} \\ &\le \frac{1+\varepsilon^2}{DM} - \frac{1}{DM} = \frac{\varepsilon^2}{DM}. \end{aligned}$$

Proof of (3): Recalling that the statistical difference between two random variables X and Y is equal to $\frac{1}{2}|X - Y|_1$, we have:

$$\begin{aligned} \Delta((H, H(X)), U_d \times U_m) &= \frac{1}{2} \cdot |(H, H(X)) - U_d \times U_m|_1 \\ &\le \frac{\sqrt{DM}}{2} \cdot \|(H, H(X)) - U_d \times U_m\| \\ &\le \frac{\sqrt{DM}}{2} \cdot \sqrt{\frac{\varepsilon^2}{DM}} \\ &= \frac{\varepsilon}{2}. \end{aligned}$$

Thus, we have in fact obtained a strong $(k, \varepsilon/2)$-extractor. □

The proof above actually shows that $\mathrm{Ext}(x, h) = h(x)$ extracts with respect to collision probability, or equivalently, with respect to the ℓ_2-norm. This property may be expressed in terms of Rényi entropy $\mathrm{H}_2(Z) \stackrel{\text{def}}{=} \log(1/\mathrm{CP}(Z))$. Indeed, we can define $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ to be a (k, ε) *Rényi-entropy extractor* if $\mathrm{H}_2(X) \ge k$ implies $\mathrm{H}_2(\mathrm{Ext}(X, U_d)) \ge m - \varepsilon$ (or $\mathrm{H}_2(U_d, \mathrm{Ext}(X, U_d)) \ge m + d - \varepsilon$ for strong Rényi-entropy extractors). Then the above proof shows that pairwise-independent hash functions yield strong Rényi-entropy extractors.

In general, it turns out that an extractor with respect to Rényi entropy must have seed length $d \ge \min\{m/2, n - k\} - O(1)$ (as opposed to $d = O(\log n)$); this explains why the seed length in the above extractor is large. (See Problem 6.4.)

6.2.2 Extractors vs. Expanders

Extractors have a natural interpretation as graphs. Specifically, we can interpret an extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ as the neighbor function of a bipartite multigraph $G = ([N], [M], E)$ with $N = 2^n$ left-vertices, $M = 2^m$ right-vertices, and left-degree $D = 2^d$,[3] where the r'th neighbor of left-vertex u is $\mathrm{Ext}(u, r)$. Typically $n \gg m$, so the graph is very unbalanced. It turns out that the extraction property of Ext is related to various "expansion" properties of G. In this section, we explore this relationship.

Let $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ be a (k, ε)-extractor and $G = ([N], [M], E)$ the associated graph. Recall that it suffices to examine Ext with respect to *flat* k-sources: in this case, the extractor property says that given a subset S of size $K = 2^k$ on the left, a random neighbor of a random element of S should be close to uniform on the right. In particular, if $S \subseteq [N]$ is a subset on the

[3] This connection is the reason we use d to denote the seed length of an extractor.

left of size K, then $|N(S)| \geq (1-\varepsilon)M$. This property is just like vertex expansion, except that it ensures expansion only for sets of size exactly K, not any size $\leq K$. Recall that we call such a graph an $(=K, A)$ *vertex expander* (Definition 5.32. Indeed, this gives rise to the following weaker variant of extractors.

Definition 6.19 (dispersers). A function $\text{Disp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-*disperser* if for every k-source X on $\{0,1\}^n$, $\text{Disp}(X, U_d)$ has a support of size at least $(1-\varepsilon)\cdot 2^m$.

While extractors can be used to simulate **BPP** algorithms with a weak random source, dispersers can be used to simulate **RP** algorithms with a weak random source. (See Exercise **??**.)

Then, we have:

Proposition 6.20. Let $n, m, d \in \mathbb{N}$, $K = 2^k \in \mathbb{N}$, and $\varepsilon > 0$. A function $\text{Disp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-disperser iff the corresponding bipartite multigraph $G = ([N], [M], E)$ with left-degree D is an $(=K, A)$ vertex expander for $A = (1-\varepsilon)\cdot M/K$.

Note that extractors and dispersers are interesting even when $M \ll K$, so the expansion parameter A may be less than 1. Indeed, $A < 1$ is interesting for vertex "expanders" when the graph is highly imbalanced. Still, for an *optimal* extractor, we have $M = \Theta(\varepsilon^2 KD)$ (because $m = k + d - 2\log(1/\varepsilon) - \Theta(1)$), which corresponds to expansion factor $A = \Theta(\varepsilon^2 D)$. (An optimal disperser actually gives $A = \Theta(D/\log(1/\varepsilon))$.) Note this is smaller than the expansion factor of $D/2$ in Ramanujan graphs and $D - O(1)$ in random graphs; the reason is that those expansion factors are for "small" sets, whereas here we are asking for sets to expand to almost the entire right-hand side.

Now let's look for a graph-theoretic property that is *equivalent* to the extraction property. Ext is a (k,ε)-extractor iff for every set $S \subseteq [N]$ of size K,

$$\Delta(\text{Ext}(U_S, U_{[D]}), U_{[M]}) = \max_{T \subseteq [M]} \left| \Pr\left[\text{Ext}(U_S, U_{[D]}) \in T\right] - \Pr\left[U_{[M]} \in T\right] \right| \leq \varepsilon,$$

where U_S denotes the uniform distribution on S. This inequality may be expressed in graph-theoretic terms as follows. For every set $T \subseteq [M]$,

$$\begin{aligned} &\left| \Pr\left[\text{Ext}(U_S, U_{[D]}) \in T\right] - \Pr\left[U_{[M]} \in T\right] \right| \leq \varepsilon \\ \Leftrightarrow \quad &\left| \frac{|e(S,T)|}{|S|D} - \frac{|T|}{M} \right| \leq \varepsilon \\ \Leftrightarrow \quad &\left| \frac{|e(S,T)|}{ND} - \mu(S)\mu(T) \right| \leq \varepsilon\mu(S), \end{aligned}$$

where $e(S,T)$ is the set of edges from S to T.

Thus, we have:

Proposition 6.21. A function $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor iff the corresponding bipartite multigraph $G = ([N], [M], E)$ with left-degree D has the property that $||e(S,T)|/ND - \mu(S)\mu(T)| \leq \varepsilon\mu(S)$ for every $S \subseteq [N]$ of size K and every $T \subseteq [M]$.

Note that this is very similar to the Expander Mixing Lemma (Lemma 4.15), which states that if a graph G has spectral expansion λ, then for *all* sets $S, T \subseteq [N]$ we have

$$\left| \frac{|e(S,T)|}{ND} - \mu(S)\mu(T) \right| \leq \lambda \sqrt{\mu(S)\mu(T)}.$$

It follows that if $\lambda\sqrt{\mu(S)\mu(T)} \leq \varepsilon\mu(S)$ for all $S \subseteq [N]$ of size K and all $T \subseteq [N]$, then G gives rise to a (k, ε)-extractor (by turning G into a D-regular bipartite graph with N vertices on each side in the natural way). It suffices for $\lambda \leq \varepsilon \cdot \sqrt{K/N}$ for this to work.

We can use this connection to turn our explicit construction of spectral expanders into an explicit construction of extractors. To achieve $\lambda \leq \varepsilon \cdot \sqrt{K/N}$, we can take an appropriate power of a constant-degree expander. Specifically, if G_0 is a D_0-regular expander on N vertices with bounded second eigenvalue, we can consider the tth power of G_0, $G = G_0^t$, where $t = O(\log((1/\varepsilon)\sqrt{N/K})) = O(n - k + \log(1/\varepsilon))$. The degree of G is $D = D_0^t$, so $d = \log D = O(t)$. This yields the following result:

Theorem 6.22. For every $n, k \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^n$ with $d = O(n - k + \log(1/\varepsilon))$.

Note that the seed length is significantly better than in the construction from pairwise-independent hashing when k is close to n, say $k = n - o(n)$ (i.e. $K = N^{1-o(1)}$). The output length is n, which is much larger than the typical output length for extractors (usually $m \ll n$). Using a Ramanujan graph (rather than an arbitrary constant-degree expander), the seed length can be improved to $d = n - k + 2\log(1/\varepsilon) + O(1)$, which yields an optimal output length $n = k + d - 2\log(1/\varepsilon) - O(1)$.

Another way of proving Theorem 6.22 is to use the fact that a random step on an expanders decreases the ℓ_2 distance to uniform, like in the proof of the Leftover Hash Lemma. This analysis shows that we actually get a Rényi-entropy extractor; and thus explains the large seed length $d \approx n - k$.

Table 6.2.2 summarizes the main differences between "classic" expanders and extractors.

Expanders	**Extractors**
Measured by vertex or spectral expansion	Measured by min-entropy/statistical difference
Typically constant degree	Typically logarithmic or poly-logarithmic degree
All sets of size *at most* K expand	All sets of size *exactly* (or at least) K expand
Typically balanced	Typically unbalanced, bipartite graphs

Table 6.1 Differences between "classic" expanders and extractors

6.2.3 List-Decoding View of Extractors

In this section, we cast extractors into the same list-decoding framework that we used to capture list-decodable codes, samplers, and expanders (in Section 5.3). Recall that all of these objects could be syntactically described as functions $\Gamma : [N] \times [D] \to [M]$, and their properties could be captured by bounding the sizes of sets of the form $\mathrm{LIST}_\Gamma(T, \varepsilon) \stackrel{\text{def}}{=} \{x : \Pr_y[\Gamma(x, y) \in T] > \varepsilon\}$

for $T \subseteq [M]$. We also considered a generalization to functions $f : [M] \to [0,1]$ where we defined $\mathrm{LIST}_\Gamma(f,\varepsilon) \stackrel{\text{def}}{=} \mathrm{LIST}_\Gamma(f,\varepsilon) = \{x : \mathrm{E}_y[f(\Gamma(x,y))] > \varepsilon\}$.

Conveniently, an extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ already meets the syntax of a function $\Gamma : [N] \times [D] \to [M]$ (matching our convention that $N = 2^n$, $D = 2^d$, $M = 2^m$). The extraction property can be described in the list-decoding framework as follows:

Proposition 6.23. Let $\Gamma = \mathrm{Ext} : [N] \times [D] \to [M]$, let $K = 2^k \in \mathbb{N}$, and $\varepsilon \in [0,1]$.

(1) If Ext is a (k,ε) extractor, then for every $f : [M] \to [0,1]$, we have

$$|\mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)| < K.$$

(2) Suppose that for every $T \subseteq [M]$, we have

$$|\mathrm{LIST}_\Gamma(T, \mu(T) + \varepsilon)| \leq K.$$

Then Ext is a $(k + \log(1/\varepsilon), 2\varepsilon)$ extractor.

Proof. (1) Suppose for contradiction that $|\mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)| \geq K$. Let X be uniformly distributed over $\mathrm{LIST}_\Gamma(f, \mu(f) + \varepsilon)$. Then X is a k-source, and

$$\begin{aligned} \mathrm{E}[f(\mathrm{Ext}(X, U_{[D]}))] &= \mathop{\mathrm{E}}_{x \stackrel{\mathrm{R}}{\leftarrow} X}[f(\mathrm{Ext}(x, U_{[D]}))] \\ &> \mu(f) + \varepsilon \\ &= \mathrm{E}[f(U_{[M]})] + \varepsilon. \end{aligned}$$

By Problem 6.1, this implies that $\mathrm{Ext}(X, U_{[D]})$ and $U_{[M]}$ are ε-far, contradicting the hypothesis that Ext is a (k,ε) extractor.

(2) Let X be any $(k + \log(1/\varepsilon))$-source. We need to show that $\mathrm{Ext}(X, U_{[D]})$ is 2ε-close to $U_{[M]}$. That is, we need to show that for every $T \subseteq [M]$, $\Pr[\mathrm{Ext}(X, U_{[D]}) \in T] \leq \mu(T) + 2\varepsilon$. So let T be any subset of $[M]$. Then

$$\begin{aligned} &\Pr[\mathrm{Ext}(X, U_{[D]}) \in T] \\ &\quad \leq \Pr[X \in \mathrm{LIST}(T, \mu(T) + \varepsilon)] + \Pr[\mathrm{Ext}(X, U_{[D]}) \in T | X \notin \mathrm{LIST}(T, \mu(T) + \varepsilon)] \\ &\quad \leq |\mathrm{LIST}(T, \mu(T) + \varepsilon)| \cdot 2^{-(k + \log(1/\varepsilon))} + (\mu(T) + \varepsilon) \\ &\quad \leq K \cdot 2^{-(k + \log(1/\varepsilon))} + \mu(T) + \varepsilon \\ &\quad = \mu(T) + 2\varepsilon. \end{aligned}$$

□

The proposition does not give an *exact* list-decoding characterization of extractors, as the two parts are not exactly converses of each other. One difference is the extra $\log(1/\varepsilon)$ bits of entropy and the factor of 2 in ε appearing in Part 1. These are typically insignificant differences for extractors; indeed even an optimal extractor loses $\Theta(\log(1/\varepsilon))$ bits of entropy (cf. Theorem 6.14). A second difference is that Part 2 shows that extractors imply bounds on $|\mathrm{LIST}(f, \mu(f) + \varepsilon)|$ even for fractional

functions f, whereas Part 1 only requires bounds on $|\mathrm{LIST}(T, \mu(T) + \varepsilon)|$ to deduce the extractor property. This only makes the result stronger, and indeed we will utilize this below.

Notice that the conditions characterizing extractors here are *identical* to the ones characterizing averaging samplers in Proposition 5.30. Actually, the condition in Part 1 is the one characterizing averaging samplers, whereas the condition in Part 2 is the one characterizing boolean averaging samplers. Thus we have:

Corollary 6.24. Let $\mathrm{Ext} : [N] \times [D] \to [M]$ and $\mathrm{Samp} : [N] \to [M]^D$ be such that $\mathrm{Ext}(x, y) = \mathrm{Samp}(x)_y$. Then:

(1) If Ext is a (k, ε) extractor, then Samp is a $(K/N, \varepsilon)$ averaging sampler, where $K = 2^k$.
(2) If Samp is a $(K/N, \varepsilon)$ boolean averaging sampler, then Ext is a $(k+\log(1/\varepsilon), 2\varepsilon)$ extractor.
(3) If Samp is a (δ, ε) boolean averaging sampler, then Samp is a $(\delta/\varepsilon, 2\varepsilon)$ averaging sampler.

Thus, the only real difference between extractors and averaging samplers is one of perspective, and both perspectives can be useful. For example, in samplers, we measure the error probability $\delta = K/N = 2^k/2^n$, whereas in extractors we measure the min-entropy threshold k on its own. Thus, the sampler perspective can be more natural when δ is relatively large compared to $1/N$, and the extractor perspective when δ becomes quite close to $1/N$. Indeed, an extractor for min-entropy $k = o(n)$ corresponds to a sampler with error probability $\delta = 1/2^{(1-o(1))n}$, which means that each of the n bits of randomness used by the sampler reduces the error probability by almost a factor of 2!

We can now also describe the close connection between strong extractors and list-decodable codes when the alphabet size/output length is small.

Proposition 6.25. Let $\mathrm{Ext} : [N] \times [D] \to [M]$ and $\mathrm{Enc} : [N] \to [M]^D$ be such that $\mathrm{Ext}(x, y) = \mathrm{Enc}(x)_y$, and let $K = 2^k \in \mathbb{N}$.

(1) If Ext is a strong (k, ε) extractor, then Enc is $(1 - 1/M - \varepsilon, K)$ list-decodable.
(2) If Enc is $(1 - 1/M - \varepsilon, K)$ list-decodable, then Ext is a $(k + \log(1/\varepsilon), M \cdot \varepsilon)$ strong extractor.

Proof. (1) Follows from Corollaries 5.31 and 6.24.

(2) Let X be a $(k + \log(1/\varepsilon))$-source and $Y = U_{[D]}$. Then the statistical difference between $(Y, \mathrm{Ext}(X, Y))$ and $Y \times U_{[M]}$ equals

$$\begin{aligned}
\Delta((Y, \mathrm{Ext}(X, Y)), Y \times U_{[M]}) &= \mathop{\mathrm{E}}_{y \xleftarrow{R} Y} \left[\Delta(\mathrm{Ext}(X, y), U_{[M]})\right] \\
&= \mathop{\mathrm{E}}_{y \xleftarrow{R} Y} \left[\Delta(\mathrm{Enc}(X)_y, U_{[M]})\right] \\
&\leq \frac{M}{2} \mathop{\mathrm{E}}_{y \xleftarrow{R} Y} \left[\max_z \Pr[\mathrm{Enc}(X)_y = z] - 1/M\right]
\end{aligned}$$

where the last inequality follows from the ℓ_1 formulation of statistical difference.

So if we define $r \in [M]^D$ by setting r_y to be the value z maximizing $\Pr[\mathrm{Enc}(X)_y = z] - 1/M$, we have:

$$\begin{aligned}
\Delta((Y, \mathrm{Ext}(X,Y)), Y \times U_{[M]}) &\leq \frac{M}{2} \cdot (\Pr[(Y, \mathrm{Enc}(X)_Y) \in T_r] - 1/M)\,, \\
&\leq \frac{M}{2} \cdot (\Pr[X \in \mathrm{LIST}(T_r, 1/M + \varepsilon)] + \varepsilon) \\
&\leq \frac{M}{2} \cdot \left(2^{-(k+\log(1/\varepsilon)} \cdot K + \varepsilon\right) \\
&\leq M \cdot \varepsilon.
\end{aligned}$$

□

Note that the quantitative relationship between extractors and list-decodable codes given by Proposition **??** deteriorates extremely fast as the output length/alphabet size increases. Nevertheless, the list-decoding view of extractors as given in Proposition 6.23 turns out to be quite useful.

6.3 Constructing Extractors

In the previous sections, we have seen that very good extractors exist — extracting almost all of the min-entropy from a source with only a logarithmic seed length. But the explicit constructions we have seen (via universal hashing and spectral expanders) are still quite far from optimal in seed length, and in particular cannot be used to give a polynomial-time simulation of **BPP** with a weak random source.

Fortunately, much better extractor constructions are known — ones that extract any constant fraction of the min-entropy using a logarithmic seed length, or extract all of the min-entropy using a polylogarithmic seed length. In this section, we will see how to construct such extractors.

6.3.1 Block Sources

We introduce a useful model of sources that has more structure than an arbitrary k-source:

Definition 6.26. A random variable $X = (X_1, X_2, \ldots, X_t)$ is a $(k_1, k_2, \ldots, k_t)$ *block source* if for every $x_1, \ldots, x_{i-1}$, $X_i|_{X_1=x_1,\ldots,X_{i-1}=x_{i-1}}$ is a k_i-source. If $k_1 = k_2 = \cdots = k_t = k$, then we call X a $t \times k$ *block source.*

Note that a $(k_1, k_2, \ldots, k_t)$ block source is also a $(k_1 + \cdots + k_t)$-source, but it comes with additional structure — each block is guaranteed to contribute some min-entropy. Thus, extracting randomness from block sources is an easier task than extracting from general sources.

The study of block sources has a couple of motivations.

- They are a natural and plausible model of sources in their own right. Indeed, they are more general than unpredictable-bit sources of Section 6.1.1: if $X \in \mathrm{UnpredBits}_{n,\delta}$ is broken into t blocks of length $\ell = n/t$, then the result is a $t \times \delta'\ell$ block source, where $\delta' = \log(1/(1-\delta))$.
- We can construct extractors for general weak sources by converting a general weak source into a block source. We will see how to do this later in the lecture.

We now illustrate how extracting from block sources is easier than from general sources. The idea is that we can extract almost-uniform bits from later blocks that are essentially independent of earlier blocks, and hence use these as a seed to extract more bits from the earlier blocks. Specifically, for the case of two blocks we have the following:

Lemma 6.27. Let $\mathrm{Ext}_1 : \{0,1\}^{n_1} \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ be a (k_1, ε_1)-extractor, and $\mathrm{Ext}_2 : \{0,1\}^{n_2} \times \{0,1\}^{d_2} \to \{0,1\}^{m_2}$ be a (k_2, ε_2)-extractor with $m_2 \geq d_1$. Define $\mathrm{Ext}'((x_1, x_2), y_2) = (\mathrm{Ext}_1(x_1, y_1), z_2)$, where (y_1, z_2) is obtained by partitioning $\mathrm{Ext}_2(x_2, y_2)$ into a prefix y_1 of length d_1 and a suffix z_2 of length $m_2 - d_1$.

Then for every (k_1, k_2) block source $X = (X_1, X_2)$ taking values in $\{0,1\}^{n_1} \times \{0,1\}^{n_2}$, it holds that $\mathrm{Ext}'(X, U_{d_2})$ is $(\varepsilon_1 + \varepsilon_2)$-close to $U_{m_1} \times U_{m_2 - d_1}$.

Proof. Since X_2 is a k_2-source conditioned on any value of X_1 and Ext_2 is a (k_2, ε_2)-extractor, it follows that $(X_1, Y_1, Z_2) = (X_1, \mathrm{Ext}_2(X_2, U_{d_2}))$ is ε_2-close to $(X_1, U_{m_2}) = (X_1, U_{d_1}, U_{m_2 - d_1})$.

Thus, $(\mathrm{Ext}_1(X_1, Y_1), Z_2)$ is ε_2-close to $(\mathrm{Ext}_1(X_1, U_{d_1}), U_{m_2 - d_1})$, which is ε_1-close to $(U_{m_1}, U_{m_2 - d_1})$ because X_1 is a k_1-source and Ext_1 is a (k_1, ε_1)-extractor.

By the triangle inequality, $\mathrm{Ext}'(X, U_{d_2}) = (\mathrm{Ext}_1(X_1, Y_1), Z_2)$ is $(\varepsilon_1 + \varepsilon_2)$-close to $(U_{m_1}, U_{m_2 - d_1})$. □

The benefit of this composition is that the seed length of Ext' depends only one of the extractors (namely Ext_2) rather than being the sum of the seed lengths. (If this is reminiscent of the zig-zag product, it is because they are closely related — see Section 6.3.5). Thus, we get to extract from multiple blocks at the "price of one." Moreover, since we can take $d_1 = m_2$, which is typically much larger than d_2, the seed length of Ext' can even be smaller than that of Ext_1.

The lemma extends naturally to extracting from many blocks:

Lemma 6.28. For $i = 1, \ldots, t$, let $\mathrm{Ext}_i : \{0,1\}^{n_i} \times \{0,1\}^{d_i} \to \{0,1\}^{m_i}$ be a (k_i, ε_i)-extractor, and suppose that $m_i \geq d_{i-1}$ for every $i = 1, \ldots, t$, where we define $d_0 = 0$. Define $\mathrm{Ext}'((x_1, \ldots, x_t), y_t) = (z_1, \ldots, z_t)$, where for $i = t, \ldots, 1$, we inductively define (y_{i-1}, z_i) to be a partition of $\mathrm{Ext}_i(x_i, y_i)$ into a d_{i-1}-bit prefix and a $(m_i - d_{i-1})$-bit suffix.

Then for every $(k_1, \ldots, k_t)$ block source $X = (X_1, \ldots, X_t)$ taking values in $\{0,1\}^{n_1} \times \cdots \{0,1\}^{n_t}$, it holds that $\mathrm{Ext}'(X, U_{d_t})$ is ε-close to U_m for $\varepsilon = \sum_{i=1}^{t} \varepsilon_i$ and $m = \sum_{i=1}^{t} (m_i - d_{i-1})$.

We remark that this composition preserves "strongness." If each of the Ext_i's correspond to strong extractors in the sense that their seeds are prefixes of their outputs, then Ext' will also correspond to a strong extractor. If in addition $d_1 = d_2 = \cdots = d_t$, then this construction can be seen as simply using the same seed to extract from all blocks.

Already with this simple composition, we can simulate **BPP** with an unpredictable-bit source (even though deterministic extraction from such sources is impossible by Proposition 6.6). As noted above, by breaking an unpredictable-bit source X with parameter δ into blocks of length ℓ, we obtain a $t \times k$ block source for $t = n/\ell$, $k = \delta'\ell$, and $\delta' = \log(1/(1-\delta))$.

Suppose that δ is a constant. Set $\ell = (10/\delta') \log n$, so that X is a $t \times k$ block source for $k = 10 \log n$, and define $\varepsilon = n^{-2}$. Letting $\mathrm{Ext} : \{0,1\}^{\ell} \times \{0,1\}^{d} \to \{0,1\}^{d+m}$ be the (k, ε) extractor

using universal hash functions (Theorem 6.18), we have:

$$\begin{aligned} d &= O(\ell) = O(\log n) \qquad \text{and} \\ m &= k - 2\log\frac{1}{\varepsilon} - O(1) > k/2 \end{aligned}$$

Composing Ext with itself t times as in Lemma 6.27, we obtain $\mathrm{Ext}' : \{0,1\}^{t\cdot\ell} \times \{0,1\}^d \to \{0,1\}^{d+t\cdot m}$ such that $\mathrm{Ext}'(X, U_d)$ is ε'-close to uniform, for $\varepsilon' = 1/n$. (Specifically, $\mathrm{Ext}'((x_1, \ldots, x_t), h) = (h, h(x_1), \ldots, h(x_t))$.) This tells us that Ext' essentially extracts half of the min-entropy from X, given a random seed of logarithmic length. Plugging this extractor into the construction of Proposition 6.15 gives us the following result.

Theorem 6.29. For every constant $\delta > 0$, we can simulate **BPP** with an unpredictable-bit source of parameter δ. More precisely, for every $L \in \mathbf{BPP}$ and every constant $\delta > 0$, there is a polynomial-time algorithm A and a polynomial q such that for every $w \in \{0,1\}^*$ and every source $X \in \mathrm{UnpredBits}_{q(|w|),\delta}$, the probability that $A(w; X)$ errs is at most $1/|w|$.

6.3.2 Reducing General Sources to Block Sources

Given the results of the previous section, a common approach to constructing extractors for general k-sources is to reduce the case of general k-sources to that of block sources.

One approach to doing this is as follows. Given a k-source X of length n, where $k = \delta n$, pick a (pseudo)random subset S of the bits of X, and let $W = X|_S$ be the bits of X in those positions. If the set S is of size ℓ, then we expect that W will have at least roughly $\delta\ell$ bits of min-entropy (with high probability over the choice of S). Moreover, W can have at most ℓ bits of min-entropy, so if $\ell < \delta n$, intuitively there must still should be at least min-entropy left in X. (This is justified by Lemma 6.30 below.) Thus, the pair (W, X) should be a block source. This approach can be shown to work for appropriate ways of sampling the set S, and recursive applications of it was the original approach to constructing good extractors (and is still useful in various contexts today). The fact mentioned above, that conditioning on a string of length ℓ reduces min-entropy by at most ℓ bits, is given by the following lemma (which is very useful when working with min-entropy) .

Lemma 6.30 (chain rule for min-entropy). If (W, X) are two jointly distributed random variables, where (W, X) is a k-source and $|\mathrm{Supp}(W)| \leq 2^\ell$, then for every $\varepsilon > 0$, it holds that with probability at least $1 - \varepsilon$ over $w \stackrel{R}{\leftarrow} W$, $X|_{W=w}$ is a $(k - \ell - \log(1/\varepsilon))$-source.

The proof of this lemma is left as an exercise (Problem 6.1).

This is referred to as the "chain rule" for min-entropy by analogy with the chain rule for Shannon entropy, which states that $\mathrm{H}_{Sh}(X|W) = \mathrm{H}_{Sh}(W, X) - \mathrm{H}_{Sh}(W)$, where the conditional Shannon entropy is defined to be $\mathrm{H}_{Sh}(X|W) = \mathrm{E}_{w \stackrel{R}{\leftarrow} W}[\mathrm{H}_{Sh}(X|_{W=w})]$. Thus, if $\mathrm{H}_{Sh}(W, X) \geq k$ and W is of length at most ℓ, we have $\mathrm{H}_{Sh}(X|W) \geq k - \ell$. The chain rule for min-entropy is not quite as clean; we need to assume that W has small support (rather than just small min-entropy) and we lose $\log(1/\varepsilon)$ bits of additional min-entropy.

Another approach, which we will follow, is based on the observation that every source of high min-entropy rate (namely, greater than 1/2) is (close to) a block source, as shown by the lemma below. Thus, we will try to convert arbitrary sources into ones of high min-entropy rate.

Lemma 6.31. If X is an $(n-\Delta)$-source of length n, and $X = (X_1, X_2)$ is a partition of X into blocks of lengths n_1 and n_2, then for every $\varepsilon > 0$, (X_1, X_2) is ε-close to some $(n_1 - \Delta, n_2 - \Delta - \log(1/\varepsilon))$ block source.

Consider $\Delta = \alpha n$ for a constant $\alpha < 1/2$, and $n_1 = n_2 = n/2$. Then each block contributes min-entropy at least $(1/2 - \alpha)n$. The proofs of Lemmas 6.30 and 6.31 are left as exercises (Problem 6.1).

6.3.3 Condensers

The previous section left us with the problem of converting a general k-source into one of high min-entropy rate. We will do this via the following kind of object:

Definition 6.32. A function $\mathrm{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $k \to_\varepsilon k'$ *condenser* if for every k-source X on $\{0,1\}^n$, $\mathrm{Con}(X, U_d)$ is ε-close to some k'-source. Con is *lossless* if $k' = k + d$.

If $k'/m > k/n$, then the condenser increases the min-entropy rate, intuitively making extraction an easier task. Indeed, condensers with $k' = m$ are simply extractors themselves.

Like extractors, it is often useful to view condensers graph-theoretically. Specifically, we think of Con as the neighbor function of a bipartite multigraph G with $N = 2^n$ left vertices, left degree $D = 2^d$, and $M = 2^m$ right vertices.

The *lossless* condensing property of Con turns out to be equivalent to G having vertex expansion close to the degree:

Proposition 6.33. Let $n, d, m \in \mathbb{N}$, $K = 2^k \in \mathbb{N}$, and $\varepsilon > 0$. A function $\mathrm{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $k \to_\varepsilon k + d$ lossless condenser if and only if the corresponding bipartite multigraph $G = ([N], [M], E)$ of left degree D is an $(= K, (1-\varepsilon)D)$ vertex expander.

Proof. $\Rightarrow$: Let $S \subseteq [N]$ be any set of size K. Then U_S is a k-source, so $\mathrm{Con}(U_S, U_d)$ is ε-close to a $(k+d)$-source. This implies that $|\mathrm{Supp}(\mathrm{Con}(U_S, U_d))| \geq (1-\varepsilon) \cdot 2^{k+d}$. Noting that $\mathrm{Supp}(\mathrm{Con}(U_S, U_d)) = N(S)$ completes the proof.

$\Leftarrow$: By Lemma 6.10, it suffices to prove that for every subset $S \subseteq [N]$ of size K, it holds that $\mathrm{Con}(U_S, U_d)$ is ε-close to a $(k+d)$-source. By expansion, we know that $|N(S)| \geq (1-\varepsilon) \cdot KD$. Since there are only KD edges leaving S, by redirecting εKD of the edges, we can ensure that they all go to KD distinct vertices in $[M]$. The uniform distribution on these KD vertices is a $(k+d)$-source that is ε-close to $\mathrm{Con}(U_S, U_d)$. □

Recall that vertex expansion normally corresponds to the *disperser* property (see Proposition 6.20), which is weaker than extraction and condensing. Indeed, vertex expansion generally does not guarantee much about the distribution induced on a random neighbor of a set S, except that its support is large. However, in case the expansion is very close to the degree ($A = (1-\varepsilon) \cdot D$), then the distribution must be nearly flat (as noted in the above proof).

By applying Proposition 6.33 to the expanders based on Parvaresh–Vardy codes (Theorem 5.35), we get the following lossless condenser:

Theorem 6.34. For every constant $\alpha > 0$, for all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit

$$k \rightarrow_\varepsilon k + d$$

lossless condenser $\mathrm{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(\log n + \log(1/\varepsilon))$ and $m = (1 + \alpha)k + O(\log(n/\varepsilon))$.

Note that setting α to be a small constant, we obtain an output min-entropy rate arbitrarily close to 1.

Problem 6.5 gives a simple extractor Ext for high min-entropy rate when the error parameter ε is constant. Applying that extractor to the output of the above condenser, we obtain extractors with a seed length of $d = O(\log n)$ that extract $\Omega(k)$ almost-uniform bits (with constant error ε) from sources of any desired min-entropy k. In the next section, we will use the above condenser to give an efficient construction of extractors for arbitrary values of ε.

We remark that the fact that having output min-entropy rate bounded away from 1 is not inherent for lossless condensers. Nonconstructively, there exist lossless condensers with output length $m = k + d + \log(1/\varepsilon) + O(1)$, and Open Problem 5.36 about expanders can be restated in the language of lossless condensers as follows:

Open Problem 6.35. Give an explicit construction of a $k \rightarrow_\varepsilon k + d$ lossless condenser $\mathrm{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(\log n)$, $m = k + d + O(1)$, and $\varepsilon = .01$.

If we had such a condenser, then we could get extractors that extract *all* but $O(1)$ bits of the min-entropy by then applying extractors based on spectral expanders (Theorem 6.22).

6.3.4 The Extractor

In this section, we will use the ideas outlined in the previous section — namely condensing and block-source extraction — to construct an extractor that is optimal up to constant factors.

Theorem 6.36. For all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq k/2$ and $d = O(\log(n/\varepsilon))$.

We will use the following building block, constructed in Problem 6.7.

Lemma 6.37. For every *constant* $t > 0$ and all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq k/2$ and $d = k/t + O(\log(n/\varepsilon))$.

The point is that this extractor has a seed length that is an arbitrarily large constant factor (approximately $t/2$) smaller than its output length. Thus, if we use it as Ext_2 in the block-source extraction of Lemma 6.27, the resulting seed length will be smaller than that of Ext_1 by an arbitrarily large constant factor. (The seed length of the composed extractor Ext' in Lemma 6.27 is the same of that as Ext_2, which will be a constant factor smaller than its output length m_2, which we can take to be equal to the seed length d_1 of Ext_1.)

Overview of the Construction. Note that for small min-entropies k, namely $k = O(\log(n/\varepsilon))$, the extractor we want is already given by Lemma 6.37 with seed length d smaller than the output length m by any constant factor. (If we allow $d \geq m$, then extraction is trivial — just output the seed.) Thus, our goal will be to recursively construct extractors for large min-entropies using extractors for smaller min-entropies. Of course, if $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k_0, ε) extractor, say with $m = k_0/2$, then it is also a (k, ε) extractor for every $k \geq k_0$. The problem is that the output length is only $k_0/2$ rather than $k/2$. Thus, we need to increase the output length. This can be achieved by simply applying extractors for smaller min-entropies several times:

Lemma 6.38. Suppose $\mathrm{Ext}_1 : \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ is a (k_1, ε_1) extractor and $\mathrm{Ext}_2 : \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{m_2}$ is a (k_2, ε_2) extractor for $k_2 = k_1 - m_1 - \log(1/\varepsilon_3)$. Then $\mathrm{Ext}' : \{0,1\}^n \times \{0,1\}^{d_1+d_2} \to \{0,1\}^{m_1+m_2}$ defined by $\mathrm{Ext}'(x,(y_1,y_2)) = (\mathrm{Ext}(x,y_1), \mathrm{Ext}(x,y_2))$ is a $(k_1, \varepsilon_1+\varepsilon_2+\varepsilon_3)$ extractor.

The proof of this lemma follows from Lemma 6.30. After conditioning a k_1-source X on $W = \mathrm{Ext}_1(X, U_{d_1})$, X still has min-entropy at least $k_1 - m_1 - \log(1/\varepsilon_3) = k_2$ (except with probability ε_3), and thus $\mathrm{Ext}_2(X, U_{d_2})$ can extract an additional m_2 almost-uniform bits.

To see how we might apply this, consider setting $k_1 = .8k$ and $m_1 = k_1/2$, $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon \geq 2^{-.1k}$, $k_2 = k_1 - m_1 - \log(1/\varepsilon_3) \in [.3k, .4k]$, and $m_2 = k_2/2$. Then we obtain a $(k, 3\varepsilon)$ extractor Ext' with output length $m = m_1 + m_2 > k/2$ from two extractors for min-entropies k_1, k_2 that are smaller than k by a constant factor, and we can hope to construct the latter two extractors recursively via the same construction.

Now, however, the problem is that the seed length grows by a constant factor in each level of recursion (e.g. if $d_1 = d_2 = d$ in Lemma 6.38, we get seed length $2d$ rather than d). Fortunately, block source extraction using the extractor of Lemma 6.37 gives us a method to reduce the seed length by a constant factor. (The seed length of the composed extractor Ext' in Lemma 6.27 is the same of that as Ext_2, which will be a constant factor smaller than its output length m_2, which we can take to be equal to the seed length d_1 of Ext_1. Thus, the seed length of Ext' will be a constant factor smaller than that of Ext_1.) In order to apply block source extraction, we first need to convert our source to a block source; by Lemma 6.31, we can do this by using the condenser of Theorem 6.34 to make its entropy rate close to 1.

One remaining issue is that the error ε still grows by a constant factor in each level of recursion. However, we can start with polynomially small error at the base of the recursion and there are only logarithmically many levels of recursion, so we can afford this blow-up.

We now proceed with the proof details. It will be notationally convenient to do the steps in the reverse order from the description above — first we will reduce the seed length by a constant factor via block-source extraction, and then apply Lemma 6.38 to increase the output length.

Proof of Theorem 6.36. Fix $n \in \mathbb{N}$ and $\varepsilon_0 > 0$. Set $d = c\log(n/\varepsilon_0)$ for an error parameter ε_0 and a sufficiently large constant c to be determined in the proof below. (To avoid ambiguity, we will keep the dependence on c explicit throughout the proof, and all big-Oh notation hides universal constants independent of c.) For $k \in [0, n]$, let $i(k)$ be the smallest nonnegative integer i such that $k \leq 2^i \cdot 8d$. This will be the level of recursion in which we handle min-entropy k; note that $i(k) \leq \log k \leq \log n$.

For every $k \in [0, n]$, we will construct an explicit $\mathrm{Ext}_k : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{k/2}$ that is a $(k, \varepsilon_{i(k)})$ extractor, for an appropriate sequence $\varepsilon_0 \leq \varepsilon_1 \leq \varepsilon_2 \cdots$. Note that we require the seed length to remain d and the fraction of min-entropy extracted to remain $1/2$ for all values of k. The construction will be by induction on $i(k)$.

Base Case: $i(k) = 0$, i.e. $k \leq 8d$. The construction of Ext_k follows from Lemma 6.37, setting $t = 9$ and taking c to be a sufficiently large constant.

Inductive Case: We construct Ext_k for $i(k) \geq 1$ from extractors $\mathrm{Ext}_{k'}$ with $i(k') < i(k)$ as follows. Given a k-source X of length n, Ext_k works as follows.

(1) We apply the condenser of Theorem 6.34 to convert X into a source X' that is ε_0-close to a k-source of length $(9/8)k + O(\log(n/\varepsilon_0))$. This requires a seed of length $O(\log(n/\varepsilon_0))$.
(2) We divide X' into two equal-sized halves (X_1, X_2). By Lemma 6.31, (X_1, X_2) is $2\varepsilon_0$-close to a $2 \times k'$ block source for
$$k' = k/2 - k/8 - O(\log(n/\varepsilon_0)) \ .$$
Note that $i(k') < i(k)$. Since $i(k) \geq 1$, we also have $k' \geq 3d - O(\log(n/\varepsilon_0)) \geq 2d$, for a sufficiently large choice of the constant c.
(3) Now we apply block-source extraction as in Lemma 6.27. We take Ext_2 to be a $(2d, \varepsilon_0)$ extractor from Lemma 6.37 with parameter $t = 16$, which will give us $m_2 = d$ output bits using a seed of length $d_2 = (2d)/16 + O(\log(n/\varepsilon_0))$. For Ext_1, we use our recursively constructed $\mathrm{Ext}_{k'}$, which has seed length d, error $\varepsilon_{i(k')}$, and output length $k'/2 \geq k/6$ (where the latter inequality holds for a sufficiently large choice of the constant c, because $k > 8d > 8c\log(1/\varepsilon)$).

All in all, our extractor so far has seed length at most $d/8 + O(\log(n/\varepsilon_0))$, error at most $\varepsilon_{i(k)-1} + O(\varepsilon_0)$, and output length at least $k/6$. This would be sufficient for our induction except that the output length is only $k/6$ rather than $k/2$. We remedy this by applying Lemma 6.38.

With one application of the extractor above, we extract at least $m_1 = k/6$ bits of the source min-entropy. Then with another application of the extractor above for min-entropy threshold $k_2 = k - m_1 - \log(1/\varepsilon_0) = 5k/6 - \log(1/\varepsilon_0)$, by Lemma 6.38, we extract another $(5k/6 - \log(1/\varepsilon_0))/6$ bits and so on. After four applications, we have extracted all but $(5/6)^4 \cdot k + O(\log(1/\varepsilon_0)) \leq k/2$ bits of the min-entropy. Our seed length is then $4 \cdot (d/8 + O(\log(n/\varepsilon_0))) \leq d$ and the total error is $\varepsilon_{i(k)} = O(\varepsilon_{i(k)-1})$.

Solving the recurrence for the error, we get $\varepsilon_i = 2^{O(i)} \cdot \varepsilon_0 \leq \mathrm{poly}(n) \cdot \varepsilon_0$, so we can obtain error ε by setting $\varepsilon_0 = \varepsilon/\mathrm{poly}(n)$. As far as explicitness, we note that computing Ext_k consists of four evaluations of our condenser from Theorem 6.34, four evaluations of $\mathrm{Ext}_{k'}$ for values of k' such that $i(k') < (i(k) - 1)$, four evaluations of the explicit extractor from Lemma 6.37, and simple string manipulations that can be done in time $\mathrm{poly}(n, d)$. Thus, the total computation time is at most $4^{i(k)} \cdot \mathrm{poly}(n, d) = \mathrm{poly}(n, d)$. □

Repeatedly applying Lemma 6.38 using extractors from Theorem 6.36, we can extract any constant fraction of the min-entropy using a logarithmic seed length, and all the min-entropy using a polylogarithmic seed length.

Corollary 6.39. The following holds for every constant $\alpha > 0$. For every $n \in \mathbb{N}$, $k \in [0, n]$, and $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq (1-\alpha)k$ and $d = O(\log(n/\varepsilon))$.

Corollary 6.40. For every $n \in \mathbb{N}$, $k \in [0, n]$, and $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k - O(\log(1/\varepsilon))$ and $d = O(\log k \cdot \log(n/\varepsilon))$.

We remark that the above construction can be modified to yield *strong* extractors achieving the same output lengths as above (so the entropy of the seed need not be lost in Corollary 6.40).

A summary of the extractor parameters we have seen is in Table 6.2.

Method	Seed Length d	Output Length m
Optimal and Nonconstructive	$\log(n-k) + O(1)$	$k + d - O(1)$
Necessary for **BPP** Simulation	$O(\log n)$	$k^{\Omega(1)}$
Spectral Expanders	$O(n-k)$	n
Pairwise Independent Hashing	$O(n)$	$k + d - O(1)$
Corollary 6.39	$O(\log n)$	$(1-\gamma)k$, any constant $\gamma > 0$
Corollary 6.40	$O(\log^2 n)$	$k - O(1)$

Table 6.2 Parameters for some constructions of $(k, .01)$ extractors.

While Theorem 6.36 and Corollary 6.39 give extractors that are optimal up to constant factors in both the seed length and output length, it remains an important open problem to get one or both of these to be optimal to within an *additive* constants while keeping the other optimal to within a constant factor.

Open Problem 6.41. Give an explicit construction of $(k, .01)$ extractors $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log n)$ and output length $m = k + d - O(1)$.

By using the condenser of Theorem 6.34, it suffices to solve achieve the above for high min-entropy rate, e.g. $k = .99n$. Alternatively, a better condenser construction would also resolve the problem. (See Open Problem 6.35.) We note that there is a recent construction of extractors with seed length $d = O(\log n)$ and output length $m = (1 - 1/\mathrm{polylog}(n))k$ (improving the output length of $m = \Omega(k)$ of Corollary 6.39, in the case of constant or slightly subconstant error).

Open Problem 6.42. Give an explicit construction of $(k, .01)$ extractors $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = \log n + O(1)$ and $m = \Omega(k)$ (or even $m = k^{\Omega(1)}$).

One of the reasons that these open problems are significant is that, in many applications of extractors, the resulting complexity depends *exponentially* on the seed length d and/or the entropy loss $k + d - m$. (An example is the simulation of **BPP** with weak random sources given by Proposition 6.15.) Thus, additive constants in these parameters corresponds to constant multiplicative factors in complexity.

Another open problem is more aesthetic in nature. The construction of Theorem 6.36 makes use of the condenser of Theorem 6.36, the Leftover Hash Lemma (Theorem 6.18) and the composition techniques of Lemmas 6.27 and Lemma 6.38 in a somewhat complex recursion. It is of interest to have a construction that is more direct. In addition to the aesthetic appeal, such a construction would likely be more practical to implement and provide more insight into extractors. In Chapter 7, we will see a very direct construction based on a connection between extractors and pseudorandom generators, but its parameters will be somewhat worse than Theorem 6.36. Thus the following remains open:

Open Problem 6.43. Give a "direct" construction of (k, ε) extractors $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log(n/\varepsilon))$ and $m = \Omega(k)$.

6.3.5 Block-Source Extraction vs. the Zig-Zag Product

To further highlight the connections between extractors and expanders, here we describe how the block-source extraction method of Section 6.3.1 (which we used in the main extractor construction of Theorem 6.36) is closely related to the zig-zag graph product of Section 4.3.2.3 (which we used in the expander construction of Theorem 4.39).

Recall the block-source extraction method of Lemma 6.27: We define $\mathrm{Ext}' : \{0,1\}^{n_1+n_2} \times \{0,1\}^{d_2} \to \{0,1\}^{m_1}$ by $\mathrm{Ext}'((x_1, x_2), y_2) = \mathrm{Ext}_1(x_1, \mathrm{Ext}_2(x_2, y_2))$. (Here we consider the special case that $m_2 = d_1$.)

Viewing the extractors as bipartite graphs, the left-vertex set is $[N_1] \times [N_2]$ and the left-degree is D_2. A random step from a vertex $(x_1, x_2) \in [N_1] \times [N_2]$ corresponds to taking a random step from x_2 in G_2 to obtain a right-hand vertex $y_1 \in \{0,1\}^{m_2}$, which we view as an edge label y for G_1. We then move to the y'th neighbor of x_1.

This is just like the first two steps of the zig-zag graph product. Why do we need a third step in the zig-zag product? It is because of the slightly different goals in the two settings. In a (spectral) expander, we consider an arbitrary initial distribution that does not have too much (Rényi) entropy, and need to add entropy to it. In a block-source extractor, our initial distribution is constrained to be a block source (so both blocks have a certain amount of min-entropy), and our goal is to produce an almost-uniform output (even if we end up with less bits than the initial entropy).

Thus, in the zig-zag setting, we must consider the following extreme cases (that are ruled out for block sources):

- The second block has no entropy given the first. Here, the step using G_2 will add entropy, but not enough to make y_1 close to uniform. Thus, we have no guarantees on the behavior of the G_1-step, and we may lose entropy with it. For this reason, we keep track of the edge used in the G_1-step — that is, we remember b_1 such that x_1 is the b_1'th neighbor of $z_1 = \mathrm{Ext}(x_1, y_1)$. This ensures that the (edge-rotation) mapping $(x_1, y_1) \mapsto (z_1, b_1)$ is a permutation and does not lose any entropy. We can think of b_1 as a "buffer" that retains any extra entropy in (x_1, y_1) that did not get extracted into z_1. So a natural idea is to just do block source extraction, but output (z_1, b_1) rather than just z_1. However, this runs into trouble with the next case.
- The first block has no entropy but the second block is completely uniform given the first.

In this case, the G_2 step cannot add any entropy and the G_1 step does not add any entropy because it is a permutation. However, the G_1 step transfers entropy into z_1. So if we add another expander-step from b_1 at the end, we can argue that it will add entropy. This gives rise to the 3-step definition of the zig-zag product.

While we analyzed the zig-zag product with respect to spectral expansion (i.e. Rényi entropy), it is also possible to do analyze it in terms of a condenser-like definition (i.e. outputting distributions ε-close to having some min-entropy). It turns out that a variant of the zig-zag product for condensers leads to a construction of constant-degree bipartite expanders with expansion $(1-\varepsilon)\cdot D$ for the balanced ($M = N$) or nearly balanced (e.g. $M = \Omega(N)$) case. However, as mentioned in Open Problems 4.43, 4.44, and 5.36, there are still several significant open problems concerning the explicit construction of expanders with vertex expansion close to the degree, involving achieving expansion $D - O(1)$, the non-bipartite case, or achieving a near-optimal number of right-hand vertices.

6.4 Exercises

Problem 6.1. (Min-entropy and Statistical Difference)

(1) Prove that for every two random variables X and Y,

$$\Delta(X,Y) = \max_f |\operatorname{E}[f(X)] - \operatorname{E}[f(Y)]| = \frac{1}{2}\cdot |X-Y|_1,$$

where the maximum is over all $[0,1]$-valued functions f. (Hint: first identify the functions f that maximize $|\operatorname{E}[f(X)] - \operatorname{E}[f(Y)]|$.)

(2) Suppose that (W,X) are jointly distributed random variables where (W,X) is a k-source and $|\mathrm{Supp}(W)| \le 2^\ell$. Show that for every $\varepsilon > 0$, with probability at least $1-\varepsilon$ over $w \stackrel{R}{\leftarrow} W$, we have $X|_{W=w}$ is a $(k-\ell-\log(1/\varepsilon))$-source.

(3) Suppose that X is an $(n-\Delta)$-source taking values in $\{0,1\}^n$, and we let X_1 consist of the first n_1 bits of X and X_2 the remaining $n_2 = n - n_1$ bits. Show that for every $\varepsilon > 0$, (X_1, X_2) is ε-close to some $(n_1 - \Delta, n_2 - \Delta - \log(1/\varepsilon))$ block source.

Problem 6.2. (Simulating Randomized Algorithms with Weak Sources)

(1) Let $A(w;r)$ be a randomized algorithm for computing a function f using m random bits such that $A(w;U_m)$ has error probability at most $1/3$ (i.e. for every w, $\Pr[A(w;U_m) \neq f(w)] \le 1/3$) and let $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a $(k, 1/7)$-extractor. Define $A'(w;x) = \mathrm{maj}_{y\in\{0,1\}^d}\{A(w;\mathrm{Ext}(x;y)\}$ (breaking ties arbitrarily). Show that for every $(k+t)$-source X, $A'(w;X)$ has error probability at most 2^{-t}.

(2) Let $A(w;r)$ be a randomized algorithm for deciding a language L using m random bits such that $A(w;U_m)$ has one-sided error probability at most $1/2$ (i.e. if $w \in L$, then $\Pr[A(w;U_m) = 1] \ge 1/2$ and if $w \notin L$, then $\Pr[A(w;U_m) = 1] = 0$) and let $\mathrm{Disp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a $(k, 1/3)$-disperser. Show how to decide L with one-sided error at most 2^{-t} given a single sample from a $(k+t)$-source X (with no other randomness) and running A and Disp each 2^d times.

Problem 6.3. (Almost-Universal Hashing) A family $\mathcal{H}$ of functions mapping domain $[N]$ to $[M]$ is said to have *collision probability* at most δ if for every $x_1 \neq x_2 \in [N]$, we have

$$\Pr_{H \xleftarrow{R} \mathcal{H}}[H(x_1) = H(x_2)] \leq \delta.$$

$\mathcal{H}$ has is ε-*almost universal* if it has collision probability at most $(1+\varepsilon)/M$. (Note that this is a relaxation of the notion of ε-almost pairwise independence from Problem 3.4.)

(1) Show that if a family $\mathcal{H} = \{h : [N] \to [M]\}$ is ε^2-almost universal, $\mathrm{Ext}(x, h) \stackrel{\mathrm{def}}{=} h(x)$ is a (k, ε) strong extractor for $k = m + 2\log(1/\varepsilon) + O(1)$, where $m = \log M$.
(2) Use Problem 3.4 to deduce that for every $n \in \mathbb{N}$, $k \leq n$, and $\varepsilon > 0$, there is a (k, ε) strong extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(k + \log(n/\varepsilon))$ and $m = k - 2\log(1/\varepsilon) - O(1)$.
(3) Given a family $\mathcal{H}$ of functions mapping $[N]$ to $[M]$, we can obtain a code $\mathrm{Enc} : [N] \to [M]^{|\mathcal{H}|}$ by $\mathrm{Enc}(x)_h = h(x)$, and conversely. Show that $\mathcal{H}$ has collision probability at most δ iff Enc has minimum distance at least $1 - \delta$.
(4) Use the above connections and the list-decoding view of extractors (Proposition 6.23) to prove the Johnson Bound for small alphabets: if a code $\mathrm{Enc} : [N] \to [M]^{\hat{n}}$ has minimum distance at least $1 - 1/M - \gamma/M$, then it is $(1 - 1/M - \sqrt{\gamma}, O(M/\gamma))$ list-decodable.

Problem 6.4. (Rényi extractors) Call a function $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ a (k, ε) *Rényi extractor* if for every source X on $\{0,1\}^n$ of Rényi entropy at least k, it holds that $\mathrm{Ext}(X, U_d)$ has Rényi entropy at least $m - \varepsilon$.

(1) Prove that a (k, ε) Rényi extractor is also a $(k, O(\sqrt{\varepsilon}))$ extractor.
(2) Show that for every $n, k, m \in \mathbb{N}$ with $m \leq n$ and $\varepsilon > 0$, there exists a (k, ε) Rényi extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(\min\{n - k + \log(1/\varepsilon), m/2 + \log(n/\varepsilon)\})$.
(3) Show that if $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $(k, 1)$ Rényi extractor, then $d \geq \min\{n - k, m/2\} - O(1)$. (Hint: consider a k-source that is uniform over $\{x : \exists y \mathrm{Ext}(x, y) \in T\}$ for an appropriately chosen set T.)

Problem 6.5. (Extractors vs. Samplers) Use the connection between extractors and averaging samplers to do the following:

(1) Prove that for all constants $\varepsilon, \alpha > 0$, there is a constant $\beta < 1$ such that for all n, there is an explicit $(\beta n, \varepsilon)$ extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d \leq \log n$ and $m \geq (1 - \alpha)n$.

(2) Prove that for every $m \in \mathbb{N}$, $\varepsilon, \delta > 0$, there exists a (nonconstructive) (δ, ε) averaging sampler $\mathrm{Samp} : \{0,1\}^n \to (\{0,1\}^m)^t$ using $n = m + 2\log(1/\varepsilon) + \log(1/\delta) + O(1)$ random bits and $t = O(1/(\varepsilon^2\delta))$ samples.

(3) Suppose we are given a constant-error **BPP** algorithm that uses $r = r(n)$ random bits on inputs of length n. Show how, using the explicit extractor of Theorem 6.36, we can reduce its error probability to $2^{-\ell}$ using $O(r)+\ell$ random bits, for any polynomial $\ell = \ell(n)$. (Note that this improves the $r + O(\ell)$ given by expander walks for $\ell \gg r$.) Conclude that every problem in **BPP** has a randomized polynomial-time algorithm that only errs for $2^{q^{0.01}}$ choices of its $q = q(n)$ random bits.

Problem 6.6. (Extracting from Unpredictable-Bit Sources)

(1) Let X be a source taking values in $\{0,1\}^n$ such that for all x, y, $\Pr[X = x]/\Pr[X = y] \leq (1-\delta)/\delta$. Show that $X \in \mathrm{UnpredBits}_{n,\delta}$.

(2) Prove that for every function $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}$ and every $\delta > 0$, there exists a source $X \in \mathrm{UnpredBits}_{n,\delta}$ with parameter δ such that $\Pr[\mathrm{Ext}(X) = 1] \leq \delta$ or $\Pr[\mathrm{Ext}(X) = 0] \geq 1 - \delta$. (Hint: for $b \in \{0,1\}$, consider X that is uniform on $\mathrm{Ext}^{-1}(b)$ with probability $1-\delta$ and is uniform on $\mathrm{Ext}^{-1}(\bar{b})$ with probability δ.)

(3) (*) Show how to extract from sources in $\mathrm{UnpredBits}_{n,\delta}$ using a seeded extractor with a seed of *constant* length. That is, the seed length should not depend on the length n of the source, but only on the bias parameter δ and the statistical difference ε from uniform desired. The number of bits extracted should be $\Omega(\delta n)$.

Problem 6.7. (The Building-Block Extractor) Prove Lemma 6.37: Show that for every *constant* $t > 0$ and all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an *explicit* (k, ε)-extractor $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq k/2$ and $d = k/t + O(\log(n/\varepsilon))$. (Hint: convert the source into a block source with blocks of length $k/O(t) + O(\log(n/\varepsilon))$.)

Problem 6.8. (Encryption and Deterministic Extraction) A (one-time) *encryption scheme* with key length n and message length m consists of an encryption function $\mathrm{Enc}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ and a decryption function $\mathrm{Dec}\colon \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ such that $\mathrm{Dec}(k, \mathrm{Enc}(k,u)) = u$ for every $k \in \{0,1\}^n$ and $u \in \{0,1\}^m$. Let K be a random variable taking values in $\{0,1\}^n$. We say that $(\mathrm{Enc}, \mathrm{Dec})$ is *(statistically) ε-secure with respect to K* if for every two messages $u, v \in \{0,1\}^m$, we have $\Delta(\mathrm{Enc}(K,u), \mathrm{Enc}(K,v)) \leq \varepsilon$. For example, the *one-time pad*, where $n = m = \ell$ and $\mathrm{Enc}(k,u) = k \oplus u = \mathrm{Dec}(k,u)$ is 0-secure (aka perfectly secure) with respect to the uniform distribution $K = U_m$. For a class $\mathcal{C}$ of sources on $\{0,1\}^n$, we say that the encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ is *ε-secure with respect to $\mathcal{C}$* if Enc is ε-secure with respect to every $K \in \mathcal{C}$.

(1) Show that if there exists a deterministic ε-extractor $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}^m$ for $\mathcal{C}$, then there exists an 2ε-secure encryption scheme with respect to $\mathcal{C}$.

(2) Conversely, use the following steps to show that if there exists an ε-secure encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ with respect to $\mathcal{C}$, where $\mathrm{Enc}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$, then there exists a deterministic 2ε-extractor $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}^{m-2\log(1/\varepsilon)-O(1)}$ for $\mathcal{C}$, provided $m \geq \log n + 2\log(1/\varepsilon) + O(1)$.

   (a) For each fixed key $k \in \{0,1\}^n$, define a source X_k on $\{0,1\}^\ell$ by $X_k = \mathrm{Enc}(k, U_m)$, and let $\mathcal{C}'$ be the class of all these sources (i.e., $\mathcal{C}' = \{X_k : k \in \{0,1\}^n\}$). Show that there exists a deterministic ε-extractor $\mathrm{Ext}'\colon \{0,1\}^\ell \to \{0,1\}^{m-2\log(1/\varepsilon)-O(1)}$ for $\mathcal{C}'$, provided $m \geq \log n + 2\log(1/\varepsilon) + O(1)$.

   (b) Show that if Ext' is a deterministic ε-extractor for $\mathcal{C}'$ and Enc is ε-secure with respect to $\mathcal{C}$, then $\mathrm{Ext}(k) = \mathrm{Ext}'(\mathrm{Enc}(k, 0^m))$ is a deterministic 2ε-extractor for $\mathcal{C}$.

Thus, a class of sources can be used for secure encryption iff it is deterministically extractable.

Problem 6.9. (Extracting from Symbol-Fixing Sources*) A generalization of a bit-fixing source is a *symbol-fixing source* X taking values in Σ^n for some alphabet Σ, where subset of the coordinates of X are fixed and the rest are uniformly distributed and independent elements of Σ. For $\Sigma = \{0,1,2\}$ and $k \in [0,n]$, give an explicit ε-extractor $\mathrm{Ext} : \Sigma^n \to \{0,1\}^m$ for the class of of symbol-fixing sources on Σ^n with min-entropy at least k, with $m = \Omega(k)$ and $\varepsilon = 2^{-\Omega(k)}$. (Hint: use a random walk on a consistently labelled 3-regular expander graph.)

6.5 Chapter Notes and References

To be written

7

Pseudorandom Generators

7.1 Motivation and Definition

Our accomplishments in derandomization from the previous chapters include the following:

- Derandomizing specific algorithms, such as the ones for MaxCut and Undirected S-T Connectivity;
- Giving explicit (efficient, deterministic) constructions of various pseudorandom objects, such as expanders, extractors, and list-decodable codes, as well as showing various relations between them;
- Reducing the randomness needed for certain tasks, such as error reduction of randomized algorithms and sampling; and
- Simulating **BPP** with any weak random source.

However, all of these still fall short of answering our original motivating question, of whether *every* randomized algorithm can be efficiently derandomized. That is, does $\mathbf{BPP} = \mathbf{P}$?

As we have seen, one way to resolve this question in the positive is to use the following two-step process: First show that the number of random bits for any **BPP** algorithm can be reduced from $\text{poly}(n)$ to $O(\log n)$, and then eliminate the randomness entirely by enumeration.

Thus, we would like to have a function G that stretches a seed of $O(\log n)$ truly random bits into $\text{poly}(n)$ bits that "look random". Such a function is called a *pseudorandom generator.* The question is how we can formalize the requirement that the output should "look random" in such a way that (a) the output can be used in place of the truly random bits in *any* **BPP** algorithm, and (b) such a generator exists.

Some candidate definitions for "looks random" include the following:

- Information-theoretic or statistical measures: e.g., entropy, statistical difference from uniform distribution, pairwise independence. All of these fail one of the two criteria. For example, it is impossible for a deterministic function to increase entropy from $O(\log n)$

to n. And it is easy to construct algorithms that fail when run using random bits that are only guaranteed to be pairwise independent.

- Kolmogorov complexity, which is defined as follows: a string "looks random" if it is incompressible (cannot be generated by a Turing machine with a representation of length less than n). An appealing aspect of this notion is that it makes sense of the randomness in a fixed string (rather than a distribution). Unfortunately, it is not suitable for our purposes. Specifically, if the function G is computable (which we certainly want!) then all of its outputs have Kolmogorov complexity $O(\log n)$ (just hardwire the seed into the TM computing G), and hence are very compressible.
- Computational indistinguishability: this is the measure we will use. Intuitively, we say that a random variable X "looks random" if no *efficient* algorithm can distinguish X from a truly uniform random variable. Another way to look at it is as follows. Recall the definition of statistical difference:

  $$\Delta(X,Y) = \max_T |Pr[X \in T] - Pr[Y \in T]|.$$

  With computational indistinguishability, we simply restrict the max to be taken only over "efficient" statistical tests T (T's for which membership can be efficiently tested).

7.1.1 Computational Indistinguishability

~~Definition 7.1 (computational indistinguishability).~~ ~~Random variables X and Y taking values in $\{0,1\}^n$ are (t,ε) *indistinguishable* if for every *nonuniform* algorithm running in time t, we have~~

$$|\Pr[T(X)=1] - \Pr[T(Y)=1]| \leq \varepsilon$$

The left-hand side above is called also the *advantage* of T.

Recall that a nonuniform algorithm is an algorithm that may have some nonuniform advice hardwired in. If the algorithm runs in time t we require that the advice string is of length at most t. Typically, to make sense of complexity measures like running time, it is necessary to use asymptotic notions (e.g. because a Turing machine can encode a huge lookup table for inputs of any bounded size in its transition function). However, for nonuniform algorithms, we can avoid doing so by using Boolean circuits as our nonuniform model of computation. Similarly to Fact 3.11, every nonuniform Turing machine algorithm running in time $t(n)$ can be simulated by a sequence of Boolean circuit C_n of size $\tilde{O}(t(n))$ and conversely every sequence of Boolean circuits of size $s(n)$ can be simulated by a nonuniform Turing machine running in time $\tilde{O}(s(n))$. Thus, to make our notation cleaner, by "nonuniform algorithm running in time t', we mean "Boolean circuit of size t' (where the size is measured by the number of AND and OR gates in the circuit). Note also that we have not specified whether the the distinguisher is deterministic or randomized; this is because a probabilistic distinguisher achieving advantage greater than ε can be turned into a deterministic distinguisher achieving advantage greater than ε by nonuniformly fixing the randomness.

While we won't do so, it is also of interest to study computational indistinguishability and pseudorandomness against uniform algorithms.

Definition 7.2 (uniform computational indistinguishability). Let X_n, Y_n be some sequences of random variables on $\{0,1\}^n$ (or $\{0,1\}^{\mathrm{poly}(n)}$). For functions $t : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$, we say that $\{X_n\}$ and $\{Y_n\}$ are $(t(n), \varepsilon(n))$ *indistinguishable for uniform algorithms* if for all probabilistic algorithms T running in time $t(n)$, we have that

$$|\Pr[T(X_n) = 1] - \Pr[T(Y_n) = 1]| \leq \varepsilon(n)$$

for all sufficiently large n, where the probabilities are taken over X_n, Y_n and the random coin tosses of T.

We will focus on the nonuniform definition, but will mention results about the uniform definition as well.

7.1.2 Pseudorandom Generators

Definition 7.3. A deterministic function $G : \{0,1\}^\ell \to \{0,1\}^n$ is a (t, ε) *pseudorandom generator (PRG)* if

(1) $\ell < n$, and
(2) $G(U_\ell)$ and U_n are (t, ε) indistinguishable.

Also, note that we have formulated the definition with respect to nonuniform computational indistinguishability, but there is a natural uniform analogue of this definition.

People attempted to construct pseudorandom generators long before this definition was formulated. Their generators were tested against a battery of statistical tests (e.g. the number of 1's and 0's are approximately the same, the longest run is of length $O(\log n)$, etc.), but these fixed set of tests provided no guarantee that the generators will perform well in an arbitrary application (e.g. in cryptography or derandomization). Indeed, most classical constructions (e.g. linear congruential generators, as implemented in the standard C library) are known to fail in some applications.

Intuitively, the above definition guarantees that the pseudorandom bits produced by the generator are as good as truly random bits for *all* efficient purposes (where efficient means time at most t). In particular, we can use such a generator for derandomizing any algorithm of running time less than t. For the derandomization to be efficient, we will also need the generator to be efficiently computable.

Definition 7.4. We say a sequence of generators $\{G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n\}$ is *computable in time* $t(n)$ if there is a *uniform and deterministic* algorithm M such that for every $n \in \mathbb{N}$ and $y \in \{0,1\}^{\ell(n)}$, we have $M(1^n, x) = G_n(x)$ and $M(1^n, x)$ runs in time at most $t(n)$.

Note that even when though we define the pseudorandomness property of the generator with respect to nonuniform algorithms, the efficiency requirement refers to uniform algorithms. As usual, for readability, we will usually refer to a single generator $G = G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n$, with it being implicit that we are discussing a family $\{G_n\}$.

Theorem 7.5. Suppose that for all n there exists an $(n, 1/8)$ pseudorandom generator $G : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ computable in time $t(n)$. Then $\mathbf{BPP} \subseteq \bigcup_c \mathbf{DTIME}(2^{\ell(n^c)} \cdot (n^c + t(n^c)))$.

Proof. A **BPP** algorithm starts with some algorithm A taking an input x of length n and a sequence of random bits r, and then accepting or rejecting after at most n^c steps for some c. We can of course assume that the algorithm uses at most n^c random bits.

The idea will be to plug in the pseudorandom generator G_{n^c} to produce this sequence of random bits, then to use the pseudorandomness assumption to show that the algorithm will do just as well with that sequence, and then to enumerate over all possible seeds to produce a derandomization.

Claim 7.6. For every x of length n, $A(x; G_{n^c}(U_{\ell(n^c)}))$ errs with probability smaller than $1/2$.

> **Proof of claim:** Suppose that there exists some x on which $A(x; G_{n^c}(U_{\ell(n^c)}))$ errs with probability at least $1/2$. Then $T(\cdot) = A(x, \cdot)$ is a nonuniform algorithm running in time n^c that distinguishes $G_{n^c}(U_{\ell(n^c)})$ from U_{n^c} with advantage at least $1/2 - 1/3 > 1/8$. Notice that we are using x here as nonuniform advice; this is why we need the PRG to be robust against nonuniform tests. □

Now, enumerate over all seeds of length $\ell(n^c)$ and take a majority vote. There are $2^{\ell(n^c)}$ of them, and for each we have to run both G and A. □

Notice that we can afford for the generator G_n have running time $t(n) = \text{poly}(n)$ or even $t(n) = \text{poly}(n) \cdot 2^{O(\ell(n))}$ without affecting the time of the derandomization by than more than a polynomial amount. In particular, for this application, it is OK if the generator runs in more time than the tests it fools (which are time n in this theorem).

The theorem provides a mechanism to produce various different theorems, relating the existence of PRGs for certain seed lengths with the ability to derandomize. Let's look at some typical settings of parameters to see what we might imagine proving with this theorem. Assuming throughout that $t(n) \leq \text{poly}(n) \cdot 2^{O(\ell(n))}$, PRGs of various seed lengths can be used to simulate **BPP** in the following deterministic time classes (see Definition 3.1):

(1) Suppose that for every ε you can create a PRG with $\ell(n) = n^\varepsilon$. Then $\mathbf{BPP} \subseteq \bigcap_{\varepsilon>0} \mathbf{DTIME}(2^{n^\varepsilon}) \stackrel{\text{def}}{=} \mathbf{SUBEXP}$. Since we know that **SUBEXP** is a proper subset of **EXP**, this would be a nontrivial improvement on the current inclusion $\mathbf{BPP} \subseteq \mathbf{EXP}$ (Proposition 3.2).
(2) Suppose we had a PRG with $\ell(n) = \text{polylog}(n)$. Then $\mathbf{BPP} \subseteq \bigcup_c \mathbf{DTIME}(2^{\log^c n}) \stackrel{\text{def}}{=} \tilde{\mathbf{P}}$.
(3) Suppose we had a PRG with $\ell(n) = O(\log n)$. Then $\mathbf{BPP} = \mathbf{P}$.

Of course, all of these derandomizations are contingent on the question of whether PRGs exist. As usual, our first answer is yes but the proof is not very helpful—it is nonconstructive and thus does not provide for an efficiently computable PRG.

Proposition 7.7. For all $t \in \mathbb{N}$ and $\varepsilon > 0$, there exists a (t, ε) pseudorandom generator $G : \{0,1\}^{\ell} \to \{0,1\}^{t}$ with seed length $O(\log t + \log(1/\varepsilon))$.

Proof. The proof is by the probabilistic method. Choose $G : \{0,1\}^{\ell} \to \{0,1\}^{n}$ at random. Now, fix a time t algorithm, T. The probability (over choice of G) that T distinguishes $G(U_\ell)$ from U_n with advantage ε is at most $2^{-\Omega(2^\ell \varepsilon^2)}$, by a Chernoff bound argument. There are $2^{\mathrm{poly}(t)}$ nonuniform algorithms running in time t (i.e. circuits of size t). Thus, union-bounding over all possible T, and setting $\varepsilon = 1/t$, we get that the probability that there exists a T breaking G is at most $2^{\mathrm{poly}(t)} 2^{-\Omega(2^\ell \varepsilon^2)}$, which is less than 1 for ℓ being $O(\log t + \log(1/\varepsilon))$. □

Note that putting together Proposition 7.7 and Theorem 7.5 gives us another way to prove that **BPP** $\subseteq$ **P/poly** (Corollary 3.12). Just let the advice string be the truth table of the PRG for the proper length, and then one can use that PRG and the proof of Theorem 7.5 to derandomize **BPP**. However, if you unfold both this proof and our previous proof (where we do error reduction and then fix the coin tosses), you will see that both proofs amount to exactly the same "construction".

7.2 Cryptographic PRGs

The theory of computational pseudorandomness developed in this chapter emerged from cryptography, where researchers sought a definition that would ensure that using pseudorandom bits instead of truly random bits (e.g. when encrypting a message) would retain security against all computationally feasible attacks. In this setting, the generator G is used by the honest parties and thus should be very efficient to compute. On the other hand, the distinguisher T corresponds to an attack carried about by an adversary, and we want to protect against adversaries that invest a lot of computational resources into trying to break the system. Thus, one is led to require that the pseudorandom generators be secure even against adversaries with greater running time. The most common setting of parameters in the theoretical literature is that the generator should run in a fixed polynomial time, but the adversary can run in an arbitrary polynomial time.

Definition 7.8. A generator $G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^{n}$ is a *cryptographic pseudorandom generator* if

- G_n is computable in polynomial time. That is, there is a constant c such that G_n is computable in time n^c.
- G_n is an $(n^{\omega(1)}, 1/n^{\omega(1)})$ PRG. That is, for every constant d, G_n is an $(n^d, 1/n^d)$ pseudorandom generator for all sufficiently large n.

Due to time constraints and the fact that such generators are covered in other texts (see the Chapter Notes and References), we will not do an in-depth study of cryptographic generators, but just survey what is known about them.

The first question to ask is whether such generators exist at all. It is not hard to show that cryptographic pseudorandom generators cannot exist unless **P** $\neq$ **NP**, indeed unless **NP** $\not\subseteq$ **P/poly**. Thus, we do not expect to establish the existence of such generators unconditionally, and instead

need to make some complexity assumption. While it would be wonderful to show that **NP** $\not\subseteq$ **P/poly** implies that existence of cryptographic pseudorandom generators, that too seems out of reach. However, we can base them on the very plausible assumption that there are functions that are easy to evaluate but hard to invert.

Definition 7.9. $f_n : \{0,1\}^n \to \{0,1\}^n$ is a *one-way function* if:

(1) There is a constant c such that f is computable in time n^c.
(2) For every constant d and every nonuniform algorithm A running in time n^d:

$$\Pr[A(f(U_n)) \in f^{-1}(f(U_n))] \leq \frac{1}{n^d}$$

for all sufficiently large n.

Assuming the existence of one-way functions seems stronger than the assumption **NP** $\not\subseteq$ **BPP**. For example, it is an average-case complexity assumption, as it requires that f is hard to invert when evaluated on *random* inputs. Nevertheless, there are a number of candidate functions believed to be one-way. The simplest is integer multiplication: $f_n(x,y) = x \cdot y$, where x and y are $n/2$-bit numbers. Inverting this function is equivalent to the integer factorization problem, for which no efficient algorithm is known.

A classic and celebrated result in the foundations of cryptography is that cryptographic pseudorandom generators can be constructed from any one-way function:

Theorem 7.10. The following are equivalent:

(1) One-way functions exist.
(2) Cryptographic pseudorandom generators exist with seed length $\ell(n) = n - 1$.
(3) For every constant $\varepsilon > 0$, there exist cryptographic pseudorandom generators with seed length $\ell(n) = n^\varepsilon$.

Corollary 7.11. If one-way functions exist, then **BPP** $\subseteq$ **SUBEXP**.

What about getting a better derandomization? The proof of the above theorem is more general quantitatively. It takes any one-way function $f_\ell : \{0,1\}^\ell \to \{0,1\}^\ell$ and a parameter m, and constructs a generator $G_m : \{0,1\}^{\text{poly}(\ell)} \to \{0,1\}^m$. The proof that G_m is pseudorandom is proven by a reduction as follows. Given any algorithm T that runs in time t and distinguishes G_m from uniform with advantage ε, we construct an algorithm T' running in time $t' = t \cdot (m/\varepsilon)^{O(1)}$ inverting f_ℓ (say with probability 1/2).

Thus if f_ℓ is hard to invert by algorithms running in time $s(\ell)$, we can set $t = m = 1/\varepsilon = s(\ell)^{1/c}$ for a constant c. That is, viewing the seed length ℓ' of G_m as a function of m, we have $\ell'(m) = \text{poly}(s^{-1}(m^c))$.

Thus:

- If $s(\ell)$ can be taken to be an arbitrarily large polynomial (as the definition of one-way function above), we get seed length $\ell'(m) = m^{\varepsilon}$ and $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$ (as discussed above).
- If $s(\ell) = 2^{\ell^{\Omega(1)}}$ (as is plausible for the factoring one-way function), then we get seed length $\ell'(m) = \text{poly}(\log m)$ and $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$.

But we cannot get seed length $\ell'(m) = O(\log m)$, as needed for concluding $\mathbf{BPP} = \mathbf{P}$, from this result. Even for the maximum possible hardness $s(\ell) = 2^{\Omega(\ell)}$, we get $\ell'(m) = \text{poly}(\log m)$. In fact, Problem **??** shows that it is impossible to have a cryptographic PRG with seed length $O(\log m)$ meeting Definition **??**, where we require that G_m be pseudorandom against all poly(m)-time algorithms. However, for derandomization we only need G_m to be pseudorandom against a fixed poly-time algorithm, e.g. running in time $t = m$, and we would get such generators with seed length $O(\log m)$ if the above construction could be improved to yield seed length $\ell' = O(\ell)$ instead of $\ell' = \text{poly}(\ell)$.

Open Problem 7.12. Given a one-way function $f : \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ that is hard to invert by algorithms running in time $s = 2^{\Omega(\ell)}$, is it possible to construct a poly(m)-time computable $(m, 1/8)$ pseudorandom generator $G : \{0,1\}^{\ell'} \to \{0,1\}^{m}$ with seed length $\ell' = O(\ell)$ and output length $m = 2^{\Omega}(\ell)$?

It is known how to do this from any one-way *permutation* $f : \{0,1\}^{\ell} \to \{0,1\}^{\ell}$. In fact, the construction of pseudorandom generators from one-way permutations has a particularly simple description:

$$G_m(x, r) = (\langle x, r\rangle, \langle f(x), r\rangle, \langle f(f(x)), r\rangle, \ldots, \langle f^{(m-1)}(x), r\rangle),$$

where $|r| = |x| = \ell$ and $\langle \cdot, \cdot \rangle$ denotes inner product modulo 2. One intuition for this construction is the following. Consider the sequence $(f^{(m-1)}(U_n), f^{(m-2)}(U_n), \ldots, f(U_n), U_n)$. By the fact that f is hard to invert (but easy to evaluate) it can be argued that the $i+1$'st component of this sequence is infeasible to predict from the first i components except with negligible probability. Thus, it is the computational analogue of a block source. The pseudorandom generator then is obtained by a computational analogue of block-source extraction, using the strong extractor $\text{Ext}(x, r) = \langle x, r\rangle$. The fact that the extraction works in the computational setting, however, is much more delicate and complex to prove than in the setting of extractors, and relies on a "local list-decoding algorithm" for the corresponding (Hadamard) code. (We will discuss local list decoding in Section **??**.)

Pseudorandom Functions. It turns out that a cryptographic pseudorandom generator can be used to build an even more powerful object — a family of *pseudorandom functions*. This is a family of functions $\{f_s : \{0,1\}^{\ell} \to \{0,1\}\}_{s \in \{0,1\}^{\ell}}$ such that (a) given the seed s, the function f_s can be evaluated in polynomial time, but (b) without the seed, it is infeasible to distinguish an oracle for f_s from an oracle to a truly random function. Thus in some sense, the ℓ-bit truly random seed s is stretched to 2^{ℓ} pseudorandom bits (namely the truth table of f_s)!

Pseudorandom functions have applications in several domains:

- Cryptography

When two parties share a seed s to a PRF, they effectively share a random function $f : \{0,1\}^\ell \to \{0,1\}$ (by definition, the function they share is indistinguishable from random by any poly-time 3rd party). Thus, in order for one party to send an encrypted message m to the other, they can simply choose a random $r \xleftarrow{R} \{0,1\}^\ell$, and send $(r, f_s(r) \oplus m)$. With knowledge of s, decryption is easy; simply calculate $f_s(r)$ and XOR it to the second part of the received message. However, the value $f_s(r) \oplus m$ would look essentially random to anyone without knowledge of s.

This is just one example; pseudorandom functions have vast applicability in cryptography.

- Learning Theory
  Here, PRFs are used mainly to prove negative results. The basic paradigm in computational learning theory is that we are given a list of examples of a function's behavior, $(x_1, f(x_2)), (x_2, f(x_2)), ..., (x_k, f(x_k)))$, and we would like to predict what the function's value will be on a new data point x_{k+1} coming from the same distribution. Information-theoretically, it should be possible to predict after a small number of samples assuming that the function has a small description (e.g. is computable by a poly-sized circuit). However, essentially by definition, it should be computationally hard to predict the output of PRFs. Thus, PRFs provide examples of functions that are efficiently computable yet hard to learn (even with membership queries).
- Hardness of Proving Circuit Lower Bounds.
  One main approach to proving $\mathbf{P} \neq \mathbf{NP}$ is to show that some $f \in \mathbf{NP}$ doesn't have polynomial size circuits (equivalently, $\mathbf{NP} \not\subseteq \mathbf{P/poly}$). This approach has had very limited success- the only superpolynomial lower bounds that have been achieved have been using very restricted classes of circuits (monotone circuits, constant depth circuits, etc). For general circuits, the best lower bound that has been achieved for a problem in **NP** is roughly $4.5n$.
  Pseudorandom functions have been used to help explain why existing lower-bound techniques have so far not yielded superpolynomial circuit lower bounds. Specifically, it has been shown that any sufficiently "constructive" proof of superpolynomial circuit lower bounds (one that would allow us to certify that a randomly chosen function has no small circuits) could be used to distinguish a pseudorandom function from truly random in subexponential time and thus invert any one-way function in subexponential time.

7.3 Hybrid Arguments

In this section, we introduce a very useful proof method for working with computational indistinguishability, known as the *hybrid argument*. We use it to establish two important facts — that computational indistinguishability is preserved under taking multiple samples, and that pseudorandomness is equivalent to next-bit unpredictability.

7.3.1 Indistinguishability of Multiple Samples

The following proposition illustrates that computational indistinguishability behaves like statistical difference when taking many independent repetitions; the distance ε multiplies by the number of copies. Proving it will introduce useful techniques for reasoning about computational indistinguishability, and will also illustrate how working with such computational notions can be more

subtle than working with statistical notions.

Proposition 7.13. If X and Y are (t, ε) indistinguishable, then for every k, X^k and Y^k are $(t, k\varepsilon)$ indistinguishable (where X^k represents k independent copies of X).

Proof. We will prove the contrapositive: if there is an efficient algorithm T distinguishing X^k and Y^k with advantage greater than $k\varepsilon$, then there is an efficient algorithm T' distinguishing X and Y with advantage greater than ε. The algorithm T' will naturally use the algorithm T as a subroutine. Thus this is a *reduction* in the same spirit as reductions used elsewhere in complexity theory (**NP**-completeness). The difference in this proof from the corresponding result about statistical difference is that we need to preserve efficiency when going from T to T'.

Suppose that there exists a nonuniform algorithm T such that

$$\left|\Pr[T(X^k) = 1] - \Pr[T(Y^k) = 1]\right| > k\varepsilon \tag{7.1}$$

We can drop the absolute value in the above expression without loss of generality. (Otherwise we can replace T with its negation; recall that negations are free in our measure of circuit size.)

Now we will use a "hybrid argument." Consider the hybrid distributions $H_i = X^{k-i}Y^i$, for $i = 0, \ldots, k$. Note that $H_0 = X^k$ and $H_k = Y^k$.

Then Inequality (7.1) is equivalent to

$$\sum_{i=1}^{k} \Pr[T(H_{i-1}) = 1] - \Pr[T(H_i) = 1] > k\varepsilon,$$

meaning that there exists some i such that $\Pr[T(H_{i-1}) = 1] - \Pr[T(H_i) = 1] > \varepsilon$. The latter simply says that

$$\Pr[T(X^{k-i}XY^{i-1}) = 1] - \Pr[T(X^{k-i}YY^{i-1}) = 1] > \varepsilon.$$

By averaging, there exists some $x_1, \ldots x_{k-i}$ and $y_{k-i+2}, \ldots y_k$ such that

$$\Pr[T(x_1, \ldots x_{k-i}, X, y_{k-i+2}, \ldots y_k) = 1] - \Pr[T(x_1, \ldots x_{k-i}, Y, y_{k-i+2}, \ldots y_k) = 1] > \varepsilon.$$

Then, define $T'(z) = T(x_1, \ldots x_{k-i}, z, y_{k-i+2}, \ldots, y_k)$. Note that T' is a nonuniform algorithm with advice i, $x_1, \ldots, x_{k-i}$, $y_{k-i+2}, \ldots y_k$ hardwired in. Hardwiring these things actually costs nothing in terms of circuit size (because constant inputs can be propagated through the circuit, only eliminating gates). Thus T' is a time t algorithm such that

$$\Pr[T'(X) = 1] - \Pr[T'(Y) = 1] > \varepsilon,$$

contradicting the indistinguishability of X and Y. □

While the parameters in the above result seem to behave nicely, with (t, ε) going to $(t, k\varepsilon)$, it is actually more costly than the corresponding result for statistical difference. First, the amount of nonuniform advice used by T' is larger than that used by T. This is hidden by the fact that we are using the same measure t (namely circuit size) to bound both the time and the advice length. Second, the result is meaningless for large values of k (e.g. $k = t$), because a time t algorithm cannot read more than t bits of the input distribution X^k and Y^k.

We note that there is an analogue of the above result for computational indistinguishability against *uniform* algorithms (Definition 7.2), but it is more delicate, because we cannot simply hardwire i, $x_1, \ldots, x_{k-i}$, $y_{k-i+2}, \ldots, y_k$ as advice. Indeed, the proposition as stated is known to be false. We need to add the additional condition that the distributions X and Y are efficiently samplable. Then T' can choose $i \xleftarrow{R} [k]$ at random, and randomly sample $x_1, \ldots, x_{k-i} \xleftarrow{R} X$, $y_{k-i+2}, \ldots, y_k \xleftarrow{R} Y$.

7.3.2 Next-Bit Unpredictability

In analyzing the pseudorandom generators that we construct, it will be useful to work with a reformulation of the pseudorandomness property, which says that, given a prefix of the output, it should be hard to predict the next bit.

For notational convenience, we deviate from our usual conventions use X to refer to an r.v. on $\{0,1\}^n$ which is part of an ensemble, and we use X_i for some $i \in [n] = \{1, \ldots, n\}$ to denote the ith bit of X. We have:

Definition 7.14. Let X be a random variable distributed on $\{0,1\}^n$. For $t \in \mathbb{N}$ and $\varepsilon \in [0,1]$, we say that X is (t, ε) *next-bit unpredictable* if for every nonuniform probabilistic algorithm P running in time $t(n)$ and every $i \in [n]$, we have:

$$\Pr\left[P(X_1 X_2 \cdots X_{i-1}) = X_i\right] \leq \frac{1}{2} + \varepsilon,$$

where the probability is taken over X and the coin tosses of P.

Note that the uniform distribution $X \equiv U_n$ is $(t, 0)$ next-bit unpredictable for every t. Intuitively, if X is pseudorandom, it must be next-bit unpredictable, as this is just one specific test one can perform on X. In fact the converse also holds, and this is the direction we will use.

Proposition 7.15. Let X be a random variable distributed on $\{0,1\}^n$. If X is a (t, ε) pseudorandom, then X is $(t - O(1), \varepsilon)$ next-bit unpredictable. Conversely, if X is (t, ε) next-bit unpredictable, then it is $(t, n \cdot \varepsilon)$ pseudorandom.

Proof. Here U denotes an r.v. uniformly distributed on $\{0,1\}^n$ and U_i denotes the i'th bit of U.

pseudorandom ⇒ next-bit unpredictable. The proof is by reduction. Suppose for contradiction that X is not $(t - O(n), \varepsilon)$ next-bit unpredictable, so we have a predictor $P : \{0,1\}^{i-1} \to \{0,1\}$ that succeeds with probability at least $1/2 + \varepsilon$. We construct an algorithm $T : \{0,1\}^n \to \{0,1\}$ that distinguishes X from U_n as follows:

$$T(x_1 x_2 \cdots x_n) = \begin{cases} 1 & \text{if } P(x_1 x_2 \cdots x_{i-1}) = x_i \\ 0 & \text{otherwise.} \end{cases}$$

next-bit unpredictable ⇒ pseudorandom. Also by reduction. Suppose X is not pseudorandom, so we have a nonuniform algorithm T running in time t s.t.

$$\Pr[T(X) = 1] - \Pr[T(U) = 1] > \varepsilon,$$

where we have dropped the absolute values without loss of generality as in the proof of Proposition 7.13.

We now use a hybrid argument. Define $H_i = X_1 \circ X_2 \circ \cdots \circ X_i \circ U_{i+1} \circ U_{i+2} \circ \cdots \circ U_n$. Then $H_n = X$ and $H_0 = U$. We have:

$$\sum_{i=1}^{n} (\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1]) > \varepsilon,$$

since the sum telescopes. Thus, there must exist an i such that

$$\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1] > \varepsilon/n.$$

This says that T is more likely to output 1 when we put X_i in the i'th bit than when we put a random bit U_i. We can view U_i as being X_i with probability $1/2$ and being $\overline{X_i}$ with probability $1/2$. The only advantage T has must be coming from the latter case, because in the former case, the two distributions are identical. Formally,

$$\begin{aligned} &\Pr[T(X_1 \cdots X_{i-1} X_i U_{i+1} \cdots U_n) = 1] + 1 - \Pr[T(X_1 \cdots X_{i-1} \overline{X_i} U_{i+1} \cdots U_n) = 1] \\ &= \quad 2 \cdot (\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1]) > \frac{2\varepsilon}{n}. \end{aligned}$$

This motivates the following next-bit predictor:

$P(x_1 x_2 \cdots x_{i-1})$:

(1) Choose random bits $u_i, \ldots, u_n \stackrel{R}{\leftarrow} \{0,1\}$.
(2) Compute $b = T(x_1 \cdots x_{i-1} u_i \cdots u_n)$.
(3) If $b = 1$, output u_i, otherwise output $\overline{u_i}$.

The intuition is that T is more likely to output 1 when $u_i = x_i$ than when $u_i = \overline{x_i}$. Formally, we have:

$$\begin{aligned} &\Pr[P(X_1 \cdots X_{i-1}) = X_i] \\ &= \quad \frac{1}{2} \cdot (\Pr[T(X_1 \cdots X_{i-1} U_i U_{i+1} \cdots U_n) = 1 | U_i = X_i] + \Pr[T(X_1 \cdots X_{i-1} U_i U_{i+1} \cdots U_n) = 0 | U_i \neq X_i]) \\ &= \quad \frac{1}{2} \cdot \left(\Pr[T(X_1 \cdots X_{i-1} X_i U_{i+1} \cdots U_n) = 1] + 1 - \Pr[T(X_1 \cdots X_{i-1} \overline{X_i} U_{i+1} \cdots U_n) = 1]\right) \\ &> \quad \frac{1}{2} + \frac{\varepsilon}{n}. \end{aligned}$$

Note that as described P runs in time $t + O(n)$. Using circuit size as our measure of nonuniform time, we can reduce its running time to t as follows. First, we may nonuniformly fix the coin tosses $u_i, \ldots, u_n$ of P while preserving its advantage. Then all P does is run T on $x_1 \cdots x_{i-1}$ concatenated with some fixed bits and and either output what T does or its negation (depending on the fixed value of u_i). Fixing some input bits and negation can be done without increasing circuit size. Thus we contradict the next-bit unpredictability of X. □

We note that an analogue of this result holds for uniform distinguishers and predictors, provided that we change the definition of next-bit predictor to involve a random choice of $i \stackrel{R}{\leftarrow} [n]$ instead

of a fixed value of i, and change the time bounds in the conclusions to be $t - O(n)$ rather than $t - O(1)$ and t (we can't do tricks like in the final paragraph of the proof). In contrast to the multiple-sample indistinguishability result of Proposition 7.13, this result does not need X to be efficiently samplable for the uniform version.

7.4 Pseudorandom Generators from Average-Case Hardness

In Section 7.2, we surveyed cryptographic pseudorandom generators, which numerous applications within and outside cryptography, including to derandomizing **BPP**. However, for derandomization, we can use generators with weaker properties. Specifically, we only need $G : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ such that:

(1) G fools (nonuniform) distinguishers running in time n (as opposed to all poly(n)-time distinguishers).
(2) G is computable in time poly$(n, 2^\ell)$. In particular, *the PRG may take more time than the distinguishers it is trying to fool.*

Such a generator implies that every **BPP** algorithm can be derandomized in time poly$(n, 2^{\ell(n)})$.

The benefit of studying such generators is that we can hope to construct them under weaker assumptions than used for cryptographic generators. In particular, a generator with the properties above no longer implies $\mathbf{P} \neq \mathbf{NP}$, much less the existence of one-way functions. (Testing whether a string is an output of the generator is still an **NP** search problem, but even if we guess the seed properly, testing may take more time than the distinguishers are allowed.) However, as shown in Problem **??**, such generators still imply nonuniform circuit lower bounds for exponential time, something that is still beyond the state of the art in complexity theory. Our goal in the next couple of sections is to construct generators as above from assumptions that are as weak as possible. In this section, we will construct them from boolean functions computable in exponential time that are hard on average for nonuniform algorithms, and in the next section we will relax this to only require worst-case hardness.

7.4.1 Average-Case Hardness

A function is *hard on average* if it is hard to compute correctly on randomly chosen inputs. Formally:

Definition 7.16. For $t \in \mathbb{N}$ and $\delta \in [0,1]$, we say that a Boolean function $f : \{0,1\}^\ell \to \{0,1\}$ is (t, δ) *average-case hard* if for all nonuniform probabilistic algorithm A running in time t,

$$\Pr[A(X) = f(X)] \leq 1 - \delta.$$

Note that saying that f is (t, δ) hard for some $\delta > 0$ (possibly exponentially small) amounts to saying that f is *worst-case* hard (at least for deterministic algorithms; defining worst-case hardness for probabilistic algorithms is a bit more delicate). Thus, average-case hardness corresponds to δ that are noticeably larger than zero, e.g. $1/t^{.1}$ or constant. Indeed, in this section we will $\alpha = 1/2 - \varepsilon$ for $\varepsilon = 1/t$. That is, no efficient algorithm can compute f much better than random guessing. A typical setting of parameters we use is $t = t(\ell)$ somewhere in range from $\ell^{\omega(1)}$ (slightly superpolynomial)

to $t(\ell) = 2^{\alpha\ell}$ for a constant $\alpha > 0$. (Note that every function is computable by a nonuniform algorithm running in time roughly 2^ℓ, so we cannot take $t(\ell)$ to be any larger.) We will also require f is computable in (uniform) time $2^{O(\ell)}$ so that our pseudorandom generator will be computable in time exponential in its seed length. The existence of such an average-case hard function is quite a strong assumption, but in Section **??** we will see how to relax it to a worst-case hardness assumption.

Now we show how to obtain a pseudorandom generator from average-case hardness.

Proposition 7.17. If $f : \{0,1\}^\ell \to \{0,1\}$ is $(t, 1/2 - \varepsilon)$ average-case hard, then $G(x) = x \circ f(x)$ is a (t, ε) pseudorandom generator.

Proof. This follows from the equivalence of pseudorandomness and next-bit unpredictability. Considering uniformly random seed X, we certainly can't predict the first ℓ bits with any advantage whatsoever, so the only hope is to predict $f(x)$ from x, but f is $(\frac{1}{2} - \varepsilon)$-hard. A black-box application of Theorem **??** would lose a factor of $\ell + 1$ in the advantage ε, but we do not need to pay it here because the first ℓ bits are perfectly uniform. (Following the proof of Theorem **??**, we would have $\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1] = 0$ for $i = 1, \ldots, \ell$.) □

Note that this generator includes its seed in its output. This is impossible for cryptographic pseudorandom generators, but is feasible (as shown above) when the generator can have more resources than the distinguishers it is trying to fool.

Of course, this generator is quite weak, stretching by only one bit. We would like to get many bits out. Here are two attempts:

- Define $G(x_1 \cdots x_k) = x_1 \cdots x_k f(x_1) \cdots f(x_k)$. This is a $(t, k\varepsilon)$ pseudorandom generator because we have k independent samples of a pseudorandom distribution so nonuniform computational indistinguishability is preserved. Note that already here we are relying on *nonuniform* indistinguishability, because the distribution $(U_\ell, f(U_\ell))$ is not samplable (in time that is feasible for the distinguishers). Unfortunately, however, this construction does not improve the ratio between output length and seed length, which remains very close to 1.
- Use composition. For example, try to get two bits out using the same seed length by defining $G'(x) = G(G(x)_1 \cdots G(x)_\ell)G(x)_\ell$. This works for cryptographic pseudorandom generators, but not for the generators we are considering here. Indeed, for the generator $G(x) = xf(x)$ of Proposition 7.17, we would get $G'(x) = xf(x)f(x) \cdots f(x)$, which is clearly not pseudorandom.

7.4.2 The Nisan–Wigderson Generator

Our goal now is to show the following:

Theorem 7.18. For $t : \mathbb{N} \to \mathbb{N}$, suppose that there is a function $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(\ell)})$ [1] such that for every input length $\ell \in \mathbb{N}$, f is $(1/2 - 1/t(\ell))$-hard for nonuniform time $t(\ell)$. Then for every

[1] $\mathbf{E}$ should be contrasted with the larger class $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(\ell)})$

$m \in \mathbb{N}$, there is an $(m, 1/m)$ pseudorandom generator $G : \{0,1\}^{\ell'(m)} \to \{0,1\}^m$ with seed length $\ell'(m) = O(t^{-1}(\text{poly}(m))^2/\log m)$ that is computable in time $2^{O(\ell'(m))}$.

Note that this is similar to the seed length $\ell'(m) = \text{poly}(s^{-1}(\text{poly}(m)))$ mentioned in Section 7.2 for constructing cryptographic pseudorandom generators from one-way functions, but the average-case assumption is incomparable (and will be weakened further in the next section). In fact, it is known how to achieve a seed length $\ell(m) = O(t^{-1}(\text{poly}(m)))$, which matches what is known for constructing pseudorandom generators from one-way permutations as well as the converse implication of Problem **??**. We will not cover this improvement here, but note that for the important case of hardness $t(\ell) = 2^{\Omega(\ell)}$, we still achieve seed length $\ell(m) = O(O(\log m)^2/\log m) = O(\log m)$ and thus $\mathbf{P} = \mathbf{BPP}$. More generally, we have:

Corollary 7.19. Suppose that $\mathbf{E}$ has a $(t(\ell), 1/2 - 1/t(\ell))$ average-case hard function $f : \{0,1\}^\ell \to \{0,1\}$.

(1) If $t(\ell) = 2^{\Omega(\ell)}$, then $\mathbf{BPP} = \mathbf{P}$.
(2) If $t(\ell) = 2^{\ell^{\Omega(1)}}$, then $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$.
(3) If $t(\ell) = \ell^{\omega(1)}$, then $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$.

The idea is to apply f on *slightly dependent* inputs, i.e. x_i and x_j share very few bits. The sets of seed bits used for each output bit will be given by a *design*:

Definition 7.20. $S_1, \cdots, S_m \subseteq [d]$ is an (ℓ, a)-design if

(1) $\forall i, |S_i| = \ell$
(2) $\forall i \neq j, |S_i \cap S_j| \leq a$

We want lots of sets having small intersections over a small universe. We will use the designs established by Problem 3.2:

Lemma 7.21. For every constant $\gamma > 0$ and every $\ell, m \in \mathbb{N}$, there exists an (ℓ, a)-design $S_1, \cdots, S_m \subseteq [d]$ with $d = O\left(\frac{\ell^2}{a}\right)$ and $a = \gamma \cdot \log m$. Such a design can be constructed deterministically in time $\text{poly}(m, d)$.

Construction 7.22 (Nisan–Wigderson Generator). Given an (ℓ, a)-design $S_1, \cdots, S_m \subseteq [d]$ and a function $f : \{0,1\}^\ell \to \{0,1\}$, define the *Nisan–Wigderson generator* $G : \{0,1\}^d \to \{0,1\}^m$ as

$$G(x) = f(x|_{S_1}) f(x|_{S_2}) \cdots f(x|_{S_m})$$

where if x is a string in $\{0,1\}^d$ and $S \subseteq [d]$, $|S| = \ell$, $x|_S$ is the string of length ℓ obtained from x by selecting the bits indexed by S.

Theorem 7.23. Let $G : \{0,1\}^d \to \{0,1\}^m$ be the Nisan–Wigderson generator based on an (ℓ, a) design and a function $f : \{0,1\}^\ell \to \{0,1\}$. If f is $(1/2 - \varepsilon/m)$-hard for nonuniform time t, then G is a (t', ε) pseudorandom generator, for $t' = t - m \cdot a \cdot 2^a$.

Theorem 7.18 follows from Theorem 7.23 by setting $\varepsilon = 1/m$ and $a = \log m$, and observing that if for $\ell = t^{-1}(m^3)$, then $t' = t(\ell) - O(m \cdot a \cdot 2^a) \geq m$, so we have an $(m, 1/m)$ pseudorandom generator. The seed length is $\ell'(m) = d = O(\ell^2/\log m) = O(t^{-1}(\mathrm{poly}(m))^2/\log m)$.

Proof. Suppose G is not an (t', ε) pseudorandom generator. By Theorem **??**, there is a nonuniform time t' next-bit predictor P such that

$$\Pr[P(f(X|_{S_1})f(X|_{S_2}) \cdots f(X|_{S_{i-1}})) = f(X|_{S_i})] > \frac{1}{2} + \frac{\varepsilon}{m}, \tag{7.2}$$

for some $i \in [m]$. From P, we construct A that computes f with probability greater than $1/2 + \varepsilon/m$.

Let $Y = X|_{S_i}$. By averaging, we can fix all bits of $X|_{\bar{S}_i} = z$ such that the prediction probability is at least $1/2 + \varepsilon/m$ (over Y and the coin tosses of the predictor P). Define $f_j(y) = f(x|_{S_j})$ for $j \in \{1, \cdots, i-1\}$. (That is, $f_j(y)$ forms x by placing y in the positions in S_i and z in the others, and then applies f to $x|_{S_i}$). Then

$$\Pr_Y[P(f_1(Y) \cdots f_{i-1}(Y)) = f(Y)] > \frac{1}{2} + \frac{\varepsilon}{m}.$$

Note that $f_j(y)$ depends only on $|S_i \cap S_j| \leq a$ bits of y. Thus, we can compute each f_j with a look-up table, which we can include in the advice to our nonuniform algorithm. Indeed, every function on a bits can be computed by a boolean circuit of size at most $a \cdot 2^a$. (In fact, size at most $O(2^a/a)$ suffices.)

Then, defining $A(y) = P(f_1(y) \cdots f_{i-1}(y))$, we deduce that $A(y)$ can be computed with error probability smaller than $1/2 - \varepsilon/m$ in nonuniform time less than $t' + m \cdot a \cdot 2^a = t$. This contradicts the hardness of f. Thus, we conclude G is an (m, ε) pseudorandom generator. □

We make the following additional remarks:

(1) This is a very general construction that works for any average-case hard function f. We only used $f \in \mathbf{E}$ to deduce G is computable in $\mathbf{E}$.
(2) The reduction works for any nonuniform class of algorithms $\mathcal{C}$ where functions of logarithmically many bits can be computed efficiently.

Indeed, we will now use the same construction to obtain an *unconditional* pseudorandom generator following constant-depth circuits.

7.4.3 Derandomizing Constant-depth circuits

~~**Definition 7.24.** An *unbounded fan-in circuit* $C(x_1, \ldots, x_n)$ has input gates consisting of variables~~ x_i, their negations $\neg x_i$, and the constants 0 and 1, as well as computation gates, which can compute the AND or OR of an unbounded number of other gates (rather than just 2, as in usual Boolean circuits).[2] The *size* of such a circuit is the number of computation gates, and the *depth* is the

[2] Note that it is unnecessary to allow internal NOT gates, as these can always be pushed to the inputs via DeMorgan's Laws at no increase in size or depth.

maximum of length of a path from an input gate to the output gate.

$\mathbf{AC^0}$ is the class of functions $f : \{0,1\}^* \to \{0,1\}$ for which there exist constants c and d and a uniformly constructible sequence of unbounded fan-in circuits $(C_n)_{n\in\mathbb{N}}$ such that for all n, C_n has size at most n^c and depth at most d, and for all $x \in \{0,1\}^n$, $C_n(x) = f(x)$. $\mathbf{BPAC^0}$ defined analogously, except that C_n may have extra inputs, which are interpreted as random bits, and we we require $\Pr_r[C_n(x,r) = f(x)] \geq 2/3$.

$\mathbf{AC^0}$ is one of the richest circuit classes for which we have superpolynomial lower bounds:

Theorem 7.25. For all constant $d \in \mathbb{N}$ and every $\ell \in \mathbb{N}$, the function $\textsc{Par}_\ell : \{0,1\}^\ell \to \{0,1\}$ defined by $\textsc{Par}_\ell(x_1,\ldots,x_\ell) = \bigoplus_{i=1}^{\ell} x_i$ is $(t_d(\ell), 1/2 - 1/t_d(\ell))$-average-case hard for nonuniform unbounded fan-in circuits of depth d and size $t_d(\ell) = 2^{\Omega(\ell^{1/d})}$.

In addition to having an average-case hard function against $\mathbf{AC^0}$, we also need that $\mathbf{AC^0}$ can compute arbitrary functions on a logarithmic number of bits.

Lemma 7.26. Every function $g : \{0,1\}^a \to \{0,1\}$ can be computed by a depth 2 circuit of size 2^a.

Using the two facts with the Nisan–Wigderson pseudorandom generator construction, we obtain the following pseudorandom generator for constant-depth circuits.

Theorem 7.27. For every constant d and every m, there exists a $\mathrm{poly}(m)$-time computable $(m, 1/m)$-pseudorandom generator $G_m : \{0,1\}^{\log^{O(d)} m} \to \{0,1\}^m$ fooling nonuniform unbounded fan-in circuits of depth d (and size m).

Proof. This is proven similarly to Theorems 7.18 and 7.23, except that we take $f = \textsc{Par}_\ell$ rather than a hard function in $\mathbf{E}$, and we observe that the reduction can be implemented in a way that increases the depth by only an additive constant. Specifically, to obtain a pseudorandom generator fooling circuits of depth d, we use the hardness of $\textsc{Par}_\ell$ against unbounded fan-in circuits of depth $d' = d+2$ and size $\mathrm{poly}(m)$. Then the seed length of G $O(\ell^2/a) < O(\ell^2) = O(\log^{d'} m)^2 = \log^{O(d)} m$.

We now follow the steps of the proof of Theorem 7.18 to go from an adversary T of depth d breaking the pseudorandomness of G to a circuit A of depth d' calculating the parity function $\textsc{Par}_\ell$.

If T has depth d, then it can be verified that the next-bit predictor P constructed in the proof of Proposition 7.15 also has depth d. (Recall that negations and constants can be propagated to the inputs so they do not contribute to the depth.) Next, in the proof of Theorem 7.23, we obtain A from P by $A(y) = P(f_1(y)f_2(y)\cdots f_{i-1}(y))$ for some $i \in \{1,\ldots,m\}$ and where each f_i depends on at most a bits of y. Now we observe that A can be computed by a small constant-depth circuit (if P can). Specifically, applying Lemma **??** to each f_i, the size of A is at most $O(m \cdot 2^a) = O(m^2)$ plus the size of P and the depth of A is at most $d+2$. This contradicts the hardness of $\textsc{Par}_\ell$. □

Corollary 7.28. $\mathbf{BPAC^0} \subseteq \tilde{\mathbf{P}}$.

With more work, this can be strengthened to actually put $\mathbf{BPAC^0}$ in $\widetilde{\mathbf{AC}}^{\mathbf{0}}$. (The difficulty is that we use majority voting in the derandomization, but small constant-depth circuits cannot compute majority. However, they can compute an "approximate" majority, and this suffices.)

The above pseudorandom generator can also be used to give a quasipolynomial-time derandomization of the randomized algorithm we saw for approximately counting the number of satisfying assignments to a DNF formula (Theorem 2.34); see Problem **??**.

Improving the running time of either of these derandomizations to polynomial is an intriguing open problem.

Open Problem 7.29. Show that $\mathbf{BPAC^0} = \mathbf{AC^0}$ or even $\mathbf{BPAC^0} \subseteq \mathbf{P}$.

Open Problem 7.30 (Open Problem 2.36, restated). Give a deterministic polynomial-time algorithm for approximately counting the number of satisfying assignments to a DNF formula.

7.5 Worst-Case/Average-Case Reductions and Locally Decodable Codes

In the previous section, we saw how to construct pseudorandom generators from boolean functions that are very hard on average, where every nonuniform algorithm running in time t must err with probability greater than $1/2 - 1/t$ on a random input. Now we want to relax the assumption to refer to worst-case hardness, as captured by the following definition.

Definition 7.31. A function $f : \{0,1\}^\ell \to \{0,1\}$ is *worst-case hard for (nonuniform) time t* if, for all (nonuniform) probabilistic algorithms A running in time t, there exists $x \in \{0,1\}^\ell$ such that $\Pr[A(x) \neq f(x)] > 1/3$, where the probability is over the coin tosses of A.

Note that, for *deterministic* algorithms A, the definition simply says $\exists x\ A(x) \neq f(x)$. In the nonuniform case, restricting to deterministic algorithms is without loss of generality because we can always derandomize the algorithm using (additional) nonuniformity. Specifically, following the proof that $\mathbf{BPP} \subseteq \mathbf{P/poly}$, it can be shown that if f is worst-case hard for nonuniform deterministic algorithms running in time t, then it is worst-case hard for nonuniform probabilistic algorithms running in time t' for $t' = \Omega(t/\ell)$.

A natural goal is to be able to construct an average-case hard function from a worst-case hard function. More formally, given a function $f : \{0,1\}^\ell \to \{0,1\}$ vs. time $t = t(\ell)$, construct a function $\hat{f} : \{0,1\}^{O(\ell)} \to \{0,1\}$ such that $\hat{f}$ is average-case hard vs. time $t' = t^{\Omega(1)}$. Moreover, we would like $\hat{f}$ to be in $\mathbf{E}$ if f is in $\mathbf{E}$. (Whether we can obtain a similar result for $\mathbf{NP}$ is a major open problem, and indeed there are negative results ruling out natural approaches to doing so.)

Our approach to doing this will be via error-correcting codes. Specifically, we will show that if $\hat{f}$ is the encoding of f in an appropriate kind of error-correcting code, then worst-case hardness of f implies average-case hardness of $\hat{f}$.

Specifically, we view f as a message of length $L = 2^\ell$, and apply an error-correcting code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ to obtain $\hat{f} = \mathrm{Enc}(f)$, which we view as a function $\hat{f} : \{0,1\}^{\hat{\ell}} \to \Sigma$, where $\hat{\ell} = \log \hat{L}$. Pictorially:

$$\boxed{\text{message } f : \{0,1\}^\ell \to \{0,1\}} \longrightarrow \boxed{\text{Enc}} \longrightarrow \boxed{\text{codeword } \hat{f} : \{0,1\}^{\hat{\ell}} \to \Sigma}.$$

(Ultimately, we would like $\Sigma = \{0,1\}$, but along the way we will discuss larger alphabets.)

Now we argue the average-case hardness of $\hat{f}$ as follows. Suppose, for contradiction, that $\hat{f}$ is not δ average-case hard. By definition, there exists an efficient algorithm A with $\Pr[A(x) = \hat{f}(x)] > 1-\delta$. We may assume that A is deterministic by fixing its coins. Then A may be viewed as a received word in $\Sigma^{\hat{L}}$, and our condition on A becomes $\Delta(A, \hat{f}) < \delta$. So if Dec is a δ-decoding algorithm for Enc, then $\mathrm{Dec}(A) = f$. By assumption A is efficient, so if Dec is efficient, then f may be efficiently computed everywhere. This would contradict our worst-case hardness assumption, assuming that $\mathrm{Dec}(A)$ gives a time $t(\ell)$ algorithm for f. However, the standard notion of decoding requires reading all $2^{\hat{\ell}}$ values of the received word A and writing all 2^{ℓ} values of the message $\mathrm{Dec}(A)$, and thus $\mathrm{Time}(\mathrm{Dec}(A)) \gg 2^{\ell}$. Everything is easy for nonuniform time 2^{ℓ}, and even in the uniform case we are mostly interested in $t(\ell) \ll 2^{\ell}$. To solve this problem we introduce the notion of local decoding.

Definition 7.32. A *local δ-decoding algorithm* for some $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ is a probabilistic oracle algorithm Dec with the following property. Let $f : \{0,1\}^{\ell} \to \{0,1\}$ be any message with associated codeword $\hat{f} = \mathrm{Enc}(f)$, and let $g : \{0,1\}^{\hat{\ell}} \to \Sigma$ be such that $\Delta(g, \hat{f}) < \delta$. Then for all $x \in \{0,1\}^{\ell}$ we have $\Pr[\mathrm{Dec}^g(x) = f(x)] \geq 2/3$, where the probability is taken over the coins flips of Dec.

In other words, given oracle access to g, we want to efficiently compute any desired bit of f with high probability. So both the input (namely, g) and the output (namely, f) are treated implicitly; the decoding algorithm does not need to read/write either in its entirety. Pictorially:

This makes it possible to have sublinear-time (or even polylogarithmic-time) decoding. Also, we note that the bound of $2/3$ in the definition can be amplified in the usual way. Having formalized a notion of local decoding, we can now make our earlier intuition precise.

Proposition 7.33. Let Enc be an error-correcting code with local δ-decoding algorithm Dec that runs in time at most t_{Dec}, and let f be worst-case hard for nonuniform time t. Then $\hat{f} = \mathrm{Enc}(f)$ is (t', δ) average-case hard, where $t' = t/t_{\mathrm{Dec}}$.

Proof. We do everything as explained before except with Dec^A in place of $\mathrm{Dec}(A)$, and now the running time is at most $\mathrm{Time}(\mathrm{Dec}) \cdot \mathrm{Time}(A)$, since Dec can make at most $\mathrm{Time}(\mathrm{Dec})$ calls to A. □

We note that the reduction in this proof does not use nonuniformity in an essential way. We used nonuniformity to fix the coin tosses of A, making it deterministic. To obtain a version for uniform hardness, the coin tosses of A can be chosen randomly instead, which with high probability will not increase A's error by more than a constant factor, which we can compensate for by replacing the (t', δ) average-case hard in the conclusion with, say, $(t', \delta/3)$ average-case hard.

In light of the above proposition, our task is now to find an error-correcting code with a local decoding algorithm. Specifically, we would like the follows parameters.

(1) We want $\hat{\ell} = O(\ell)$, or equivalently $\hat{L} = \mathrm{poly}(L)$.

(2) We would like Enc to be computable in time $2^{O(\ell)} = \text{poly}(L)$. This is because we want $f \in \mathbf{E}$ to imply $\hat{f} \in \mathbf{E}$.
(3) We would like $\Sigma = \{0,1\}$ so that $\hat{f}$ is a boolean function, and have $\delta = 1/2 - \varepsilon$ so that $\hat{f}$ has sufficient average-case hardness for the pseudorandom generator construction of Theorem 7.23.
(4) Since $\hat{f}$ will be average-case hard against time $t' = t/t_{\text{Dec}}$, we would want the running time of Dec to be $t_{\text{Dec}} = \text{poly}(\ell, 1/\varepsilon)$ so that we can take $\varepsilon = t^{\Omega(1)}$ and still have $t = t^{\Omega(1)}/\text{poly}(\ell)$.

Of course, achieving $\delta = 1/2 - \varepsilon$ is not possible with our current notion of local *unique* decoding (which is only harder than the standard notion of unique decoding), and thus in the next section we will focus on getting δ to be just a fixed constant. In Section **??**, we will introduce a notion of local *list* decoding, which will enable decoding from distance $\delta = 1/2 - \varepsilon$.

In our constructions, it will be more natural to focus on the task of decoding *codeword* symbols rather than message symbols:

Definition 7.34 (locally correctible codes[3]). A *local δ-correcting algorithms* for a code $\mathcal{C} \subset \Sigma^{\hat{L}}$ is a probabilistic oracle algorithm Dec with the following property. Let $\hat{f} \in \mathcal{C}$ be any codeword, and let $g : \{0,1\}^{\hat{\ell}} \to \Sigma$ be such that $\Delta(g, \hat{f}) < \delta$. Then for all $x \in [\hat{L}]$ we have $\Pr[\text{Dec}^g(x) = \hat{f}(x)] \geq 2/3$, where the probability is taken over the coins flips of Dec.

This implies the standard definition of locally decodable codes under the (mild) constraint that the message symbols are explicitly included in the codeword.

Definition 7.35 (systematic encodings). An encoding algorithm $\text{Enc} : \{0,1\}^L \to \mathcal{C}$ for a code $\mathcal{C} \subseteq \Sigma^{\hat{L}}$ is *systematic* if there is a polynomial-time computable function $I : [L] \to [\hat{L}]$ such that for all $f \in \{0,1\}^L$, $\hat{f} = \text{Enc}(f)$, and all $x \in [L]$, we have $\hat{f}(I(x)) = f(x)$, where we interpret 0 and 1 as elements of Σ in some canonical way.

Lemma 7.36. If $\text{Enc} : \{0,1\}^L \to \mathcal{C}$ is systematic and $\mathcal{C}$ has a local δ-correcting algorithm running in time t, then Enc has a local δ-decoding algorithm (in the standard sense) running in time $t + \text{poly}(\log L)$.

Proof. If Dec_1 is the local corrector for $\mathcal{C}$ and I the mapping in the definition of systematic encoding, then $\text{Dec}_2^g(x) = \text{Dec}_1^g(I(x))$ is a local decoder for Enc. □

7.5.1 Local Decoding Algorithms

Hadamard Code. Recall the Hadamard code of message length m, which consists of the truth tables of all $\mathbb{Z}_2$-linear functions $c : \{0,1\}^m \to \{0,1\}$.

Proposition 7.37. The Hadamard code $\mathcal{C} \subseteq \{0,1\}^{2^m}$ of message length m has a local $(1/4-\varepsilon)$-correcting algorithm running in time $\mathrm{poly}(m, 1/\varepsilon)$.

Proof. We are given oracle access to $g : \{0,1\}^m \to \{0,1\}$ that is at distance less than $1/4-\varepsilon$ from some (unknown) linear function c, and we want to compute $c(x)$ at an arbitrary point $x \in \{0,1\}^m$. The idea is *random self-reducibility*: we can reduce computing c at an arbitrary point to computing c at uniformly random points, where g is likely to give the correct answer. Specifically, $c(x) = c(x \oplus r) \oplus c(r)$ for every r, and both $x \oplus r$ and r are uniformly distributed if we choose $r \overset{\text{R}}{\leftarrow} \{0,1\}^m$. The probability that g differs from c at either of these points is less than $2 \cdot (1/4-\varepsilon) = 1/2 - 2\varepsilon$. Thus $g(x \oplus r) \oplus g(r)$ gives the correct answer with probability noticeably larger than $1/2$. We can amplify this success probability by repetition. Specifically, we obtain the following local corrector:

Algorithm 7.38 (Local Corrector for Hadamard Code).
Input: an oracle $g : \{0,1\}^m \to \{0,1\}$, $x \in \{0,1\}^m$, and a parameter $\varepsilon > 0$

(1) Choose $r_1, \ldots, r_t \overset{\text{R}}{\leftarrow} \{0,1\}^m$, for $t = O(1/\varepsilon^2)$.
(2) Query $g(r_i)$ for each $i = 1, \ldots, t$.
(3) Output $\mathrm{maj}_{1 \le j \le t}\{g(r_j) \oplus g(r_j \oplus x)\}$.

If $\Delta(g, c) < 1/4 - \varepsilon$, then this algorithm will output $c(x)$ with probability at least $2/3$. □

This local decoding algorithm is optimal in terms of its decoding distance and running time, but the problem is that the Hadamard code has exponentially small rate.

Reed–Muller Code. Recall that the q-ary Reed–Muller code of degree d and dimension m consists of all multivariate polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ of total degree at most d. (Construction 5.16.) This code has minimum distance $\delta = 1 - d/q$. Reed–Muller Codes are a common generalization of both Hadamard and Reed–Solomon codes, and thus we can hope that for an appropriate setting of parameters, we will be able to get the best of both kinds of codes. That is, we want to combine the efficient local decoding of the Hadamard code with the good rate of Reed-Solomon codes.

Theorem 7.39. The q-ary Reed-Muller Code of degree d and dimension m has a local $1/12$-correcting algorithm running in time $\mathrm{poly}(m, q)$ provided $d \le q/9$ and $q \ge 36$.

Note the running time of the decoder is roughly the m'th root of the block length $\hat{L} = q^m$. When $m = 1$, our decoder can query the entire string and we simply obtain a global decoding algorithm for Reed-Solomon Codes (which we already know how to achieve from Theorem 5.19). But for large enough m, the decoder can only access a small fraction of the received word. In fact, one can improve the running time to $\mathrm{poly}(m, d, \log q)$, but the weaker result is sufficient for our purposes.

The key idea behind the decoder is to do restrictions to random *lines* in $\mathbb{F}^m$. The restriction of a Reed–Muller codeword to such a line is a Reed–Solomon codeword, and we can afford to run our global Reed–Solomon decoding algorithm on the line.

Formally, for $x, y \in \mathbb{F}^m$, we define the *line through x in direction y* as the function $\ell_{x,y} : \mathbb{F} \to \mathbb{F}^m$ given by $\ell_{x,y}(t) = x + ty$. If $g : \mathbb{F}^m \to \mathbb{F}$ is any function and $\ell : \mathbb{F} \to \mathbb{F}^m$ is a line, then we use $g|_\ell$ to denote the *restriction of g to ℓ*, which is simply the composition $g \circ \ell : \mathbb{F} \to \mathbb{F}$. Note that if p is any polynomial of total degree at most d, then $p|_\ell$ is a (univariate) polynomial of degree at most d.

So we are given an oracle g of distance less than δ from some degree d polynomial $p : \mathbb{F}^m \to \mathbb{F}$, and we want to compute $p(x)$ for some $x \in \mathbb{F}^m$. We begin by choosing a random line ℓ through x. Every point of $\mathbb{F}^m \setminus \{x\}$ lies on exactly one line through x, so the points on ℓ (except x) are distributed uniformly at random over the whole domain, and thus g and p are likely to agree on these points. Thus we can hope to use the points on this line to reconstruct the value of $p(x)$. If δ is sufficiently small compared to the degree (e.g. $\delta = 1/3(d+1)$), we can simply interpolate the value of $p(x)$ from $d+1$ random points on the line. This gives rise to the following algorithm.

Algorithm 7.40 (Local Corrector for Reed–Muller Code I).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d

(1) Choose $y \xleftarrow{R} \mathbb{F}^m$. Let $\ell = \ell_{x,y} : \mathbb{F} \to \mathbb{F}^m$ be the line through x in direction y.
(2) Query g to obtain $\beta_0 = g|_\ell(\alpha_0) = g(\ell(a_0)), ..., \beta_d = g|_\ell(\alpha_d) = g(\ell(a_d))$ where $\alpha_0, ..., \alpha_d \in \mathbb{F} \setminus \{0\}$ are any fixed points
(3) Interpolate to find a unique univariate polynomial q of degree at most d s.t. $\forall i, q(\alpha_i) = \beta_i$
(4) Output $q(0)$

Claim 7.41. If g has distance less than $\delta = 1/3(d+1)$ from some polynomial p of degree at most d, then Algorithm 7.40 will output $p(x)$ with probability greater than $2/3$.

Proof of claim: Observe that for all $\alpha_i \in \mathbb{F} \setminus \{0\}$, $\ell_{x,y}(\alpha_i)$ is uniform over the $y \in \mathbb{F}^m$. This implies that for each i,

$$\Pr_\ell[g|_\ell(\alpha_i) \neq p|_\ell(\alpha_i)] < \delta = \frac{1}{3(d+1)}.$$

By a union bound,

$$\Pr_\ell[\exists i, g|_\ell(\alpha_i) \neq p|_\ell(\alpha_i)] < (d+1) \cdot \delta = \frac{1}{3}.$$

Thus, with probability greater than $2/3$, we have $\forall i, q(\alpha_i) = p|_\ell(\alpha_i)$ and hence $q(0) = p(x)$. The running time of the algorithm is $\text{poly}(m, q)$. □

We now show how to improve the decoder to handle a larger fraction of errors, up to distance $\delta = 1/12$. We alter steps 2 and 3 in the above algorithm. In step 2, instead of querying only $d+1$ points, we query over *all* points in ℓ. In step 3, instead of interpolation, we use a *global* decoding algorithm for Reed–Solomon codes to decode the univariate polynomial $p|_\ell$. Formally, the algorithm proceeds as follows.

Algorithm 7.42 (Local Corrector for Reed–Muller Codes II).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d, where $q = |\mathbb{F}| \geq 36$ and $d \leq q/9$.

(1) Choose $y \overset{R}{\leftarrow} \mathbb{F}^m$. Let $\ell = \ell_{x,y} : \mathbb{F} \to \mathbb{F}^m$ be the line through x in direction y.
(2) Query g at all points on ℓ to obtain $g|\ell : \mathbb{F} \to \mathbb{F}$.
(3) Run the 1/3-decoder for the q-ary Reed–Solomon codes of degree d on $g|_\ell$ to obtain the (unique) polynomial q at distance less than 1/3 from $g|_\ell$ (if one exists).[4]
(4) Output $q(0)$.

Claim 7.43. If g has distance less than $\delta = 1/12$ from some polynomial p of degree at most d, and the parameters satisfy $q = |\mathbb{F}| \geq 36$, $d \leq q/9$, then Algorithm 7.42 will output $p(x)$ with probability greater than 2/3.

Proof of claim: The expected distance (between $g|_\ell$ and $p|_\ell$) is small:

$$\mathop{\mathrm{E}}_{\ell}[\Delta(g|_\ell, p|_\ell)] \leq \frac{1}{q} + \delta < \frac{1}{36} + \frac{1}{12} = \frac{1}{9},$$

where the term $\frac{1}{q}$ is due to the fact that the point x is not random. Therefore, by Markov's Inequality,

$$\Pr[\Delta(g|_\ell, p|_\ell) \geq 1/3] \leq 1/3$$

Thus, with probability at least 2/3, we have that $p|_\ell$ is the unique polynomial of degree at most d at distance less than 1/3 from $g|_\ell$ and thus q must equal $p|_\ell$. □

7.5.2 Low-Degree Extensions

Recall that to obtain locally decodable codes from locally correctible codes (as constructed above), we need to exhibit systematic encoding (Definition 7.35.) Thus, given $f\colon \{0,1\}^\ell \to \{0,1\}$, we want to encode it as a Reed–Muller codeword $\hat{f}\colon \{0,1\}^{\hat{\ell}} \to \Sigma$ s.t.:

- The encoding time is $2^{O(\ell)}$
- $\hat{\ell} = O(\ell)$
- The code is systematic in the sense of Definition 7.35. Informally, this means that f should be a "restriction" of $\hat{f}$.

Note that the usual encoding for Reed–Muller codes, where the message gives the coefficients of the polynomial, is not systematic. Instead the message should correspond to evaluations of the polynomial at certain points. Once we settle on the set of evaluation points, the task becomes one of interpolating the values at these points (given by the message) to a low-degree polynomial defined everywhere. does not suffice, since this encoding is not systematic.

The simplest approach is to use the boolean hypercube as the set of evaluation points.

[4] A 1/3-decoder for Reed–Solomon codes follows from the $(1 - 2\sqrt{d/q})$ list-decoding algorithm of Theorem 5.19. Since $1/3 \leq 1 - 2\sqrt{d/q}$, the list-decoder will produce a list containing all univariate polynomials at distance less than 1/3, and since 1/3 is smaller than half the minimum distance $(1 - d/q)$, there will be only one good decoding.

Lemma 7.44 (multilinear extension). For every $f\colon \{0,1\}^\ell \to \{0,1\}$ and every finite field $\mathbb{F}$, there exists a (unique) polynomial $\hat{f}\colon \mathbb{F}^\ell \to \mathbb{F}$ s.t. $\hat{f}_{|\{0,1\}^\ell} \equiv f$ of degree at most 1 in each variable. (and hence total degree at most ℓ).

Proof. We prove the existence of the polynomial $\hat{f}$. Define

$$\hat{f}(x_1, ..., x_\ell) = \sum_{\alpha \in \{0,1\}^\ell} f(\alpha)\delta_\alpha(x)$$

for

$$\delta_\alpha(x) = \left(\prod_{i\colon \alpha_i = 1} x_i\right)\left(\prod_{i\colon \alpha_i = 0} (1 - x_i)\right)$$

Note that for $x \in \{0,1\}^\ell$, $\delta_\alpha(x) = 1$ only when $\alpha = x$, therefore $\hat{f}_{|\{0,1\}^\ell} \equiv f$. We omit the proof of uniqueness. The bound on the individual degrees is by inspection. □

Thinking of $\hat{f}$ as an encoding of f, let's inspect the properties of this encoding.

- Since the total degree of the multilinear extension can be as large as ℓ, we need $q \geq 9\ell$ for the local corrector of Theorem 7.39 to apply.
- The encoding time is $2^{O(\ell)}$, as computing a single point of $\hat{f}$ requires summing over $2^{O(\ell)}$ elements.
- The code is systematic, since $\hat{f}$ is an extension of f.
- However, the input length is $\hat{\ell} = \ell \log q = \Theta(\ell \log \ell)$, which is slightly larger than our target of $\hat{\ell} = O(\ell)$.

To solve the problem of the input length $\hat{\ell}$ in the multi-linear encoding, we reduce the dimension of the polynomial $\hat{f}$ by changing the embedding of the domain of f: Instead of interpreting $\{0,1\}^\ell \subseteq \mathbb{F}^{\hat{\ell}}$ as an embedding of the domain of f in $\mathbb{F}^{\hat{\ell}}$, we map $\{0,1\}^\ell$ to $\mathbb{H}^m$ for some subset $\mathbb{H} \subseteq \mathbb{F}$, and as such embed it in $\mathbb{F}^m$.

More precisely, we fix a subset $\mathbb{H} \subseteq \mathbb{F}$ of size $|\mathbb{H}| = \lceil \sqrt{q} \rceil$. Choose $m = \lceil \ell / \log |\mathbb{H}| \rceil$, and fix some efficient one-to-one mapping from $\{0,1\}^\ell$ into $\mathbb{H}^m$. With this mapping, view f as a polynomial $f\colon \mathbb{H}^m \to \mathbb{F}$.

Analogously to before, we have the following.

Lemma 7.45 (low-degree extension). For every finite field $\mathbb{F}$, $\mathbb{H} \subseteq \mathbb{F}$, $m \in \mathbb{N}$, and function $f : \mathbb{H}^m \to \mathbb{F}$, there exists a (unique) $\hat{f}\colon \mathbb{F}^m \to \mathbb{F}$ of degree at most $|\mathbb{H}| - 1$ in each variable (and hence total degree at most $m \cdot (|\mathbb{H}| - 1)$) s.t. $\hat{f}|_{H^m} \equiv f$.

Using $|\mathbb{H}| = \lceil \sqrt{q} \rceil$, the total degree of $\hat{f}$ is at most $d = \ell\sqrt{q}$. So we can apply the local corrector of Theorem 7.39, as long as $q \geq 81\ell^2$ (so that $d \leq q/9$). Inspecting the properties of $\hat{f}$ as an encoding of f, we have:

- The input length is $\hat{\ell} = m \cdot \log q = \lceil \ell / \log |\mathbb{H}| \rceil \cdot \log q = O(\ell)$, as desired.
- The code is systematic as long as our mapping from $\{0,1\}^\ell$ to $\mathbb{H}^m$ is efficient.

7.5.3 Putting It Together

Combining Theorem 7.39 with Lemmas 7.45, and 7.36, we obtain the following locally decodable code:

Proposition 7.46. For every $L \in \mathbb{N}$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$, with blocklength $\hat{L} = \mathrm{poly}(L)$ and alphabet size $|\Sigma| = \mathrm{poly}(\log L)$, that has a local $(1/12)$-decoder running in time $\mathrm{poly}(\log L)$.

Using Proposition 7.33, we obtain the following conversion from worst-case hardness to average-case hardness:

Proposition 7.47. If there exists $f\colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f}\colon \{0,1\}^{O(\ell)} \to \{0,1\}^{O(\log \ell)}$ in $\mathbf{E}$ that is $(t'(\ell), 1/12)$ average-case hard for $t'(\ell) = t(\ell)/\mathit{poly}(\ell)$.

This differs from our original goal in two ways: $\hat{f}$ is not Boolean, and we only get hardness $1/12$ (instead of $1/2 - \varepsilon$). The former concern can be remedied by concatenating the code with a Hadamard code, similarly to Problem 5.2. Note that the Hadamard code is for message space $[q]$, so it can be $1/4$-decoded by brute-force in time $\mathrm{poly}(q)$ (which is the amount of time already taken by our decoder).[5] Using this, we obtain:

Theorem 7.48. For every $L \in \mathbb{N}$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\hat{L}}$ with blocklength $\hat{L} = \mathrm{poly}(L)$ that has a local $(1/48)$-decoder running in time $\mathrm{poly}(\log L)$.

Theorem 7.49. If there exists $f\colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f}\colon \{0,1\}^{O(\ell)} \to \{0,1\}$ in $\mathbf{E}$ that is $1/48$ average-case hard against (non-uniform) time $t(\ell)/\mathit{poly}(\ell)$.

An improved decoding distance can be obtained using Problem **??**.

We note that the local decoder of Theorem 7.48 not only runs in time $\mathrm{poly}(\log L)$, but also makes $\mathrm{poly}(\log L)$ queries. For some applications (such as Private Information Retrieval, see Problem **??**), it is important to have the number q of queries be as small as possible, ideally a constant. Using Reed–Muller codes of constant degree, it is possible to obtain constant-query locally decodable codes, but the blocklength will be $\hat{L} = \exp(L^{1/(q-1)})$. In a recent breakthrough, it was shown how to obtain constant-query locally decodable codes with blocklength $\hat{L} = \exp(L^{o(1)})$. Obtaining polynomial blocklength remains open.

[5] Some readers may recognize this concatenation step as the same as applying the "Goldreich–Levin hardcore predicate" to $\hat{f}$. However, for the parameters we are using (where the message space is small and we are doing unique decoding), we do not need the sophisticated Goldreich–Levin algorithm (which can be interpreted as a "local list-decoding algorithm," a notion we will define next time).

Open Problem 7.50. Are there binary codes that are locally decodable with a constant number of queries (from constant distance $\delta > 0$) and blocklength polynomial in the message length?

7.5.4 Other Connections

As shown in Problem **??**, locally decodable codes are closely related to protocols for *private information retrieval*. Another connection, and actually the setting in which these local decoding algorithms were first discovered, is to *program self-correctors*. Suppose you have a program for computing a function, such as the DETERMINANT, which happens to be a codeword in a locally decodable code (e.g. the determinant is a low-degree multivariate polynomial). Then, even if this program has some bugs and gives the wrong answer on some small fraction of inputs, you can use the local decoding algorithm to obtain the correct answer on *all* inputs with high probability.

7.6 Local List Decoding

7.6.1 Hardness Amplification

In the previous section, we saw how to use locally decodable codes to convert a worst-case hard function into one with constant average-case hardness (Theorem 7.49). Now our goal is to boost this constant hardness to $1/2 - \varepsilon$.

There are some generic techniques for doing this, known as Direct Product Theorems or the XOR Lemma (for Boolean functions). In those methods we use independent copies of the function at hand. For example, in the XOR lemma, we let f' consist of k independent copies of $\hat{f}$,

$$f'(x_1, ..., x_k) = (\hat{f}(x_1), ..., \hat{f}(x_k)).$$

Intuitively, if $\hat{f}$ is $1/12$ average-case hard, then f' should $(1-(11/12)^k)$-average case hard. Similarly, if we take the XOR of k independent copies of a Boolean function, the hardness should approach $1/2$ exponentially fast. These statements are (basically) true, though proving them turns out to be more delicate than one might expect.

The main disadvantage of this approach (for our purposes) is that the input length is $k\ell$ while we aim for input length of $O(\ell)$. To overcome this problem, it is possible to use derandomization, namely, evaluate $\hat{f}$ on *dependent* inputs instead of independent ones.

Thus, we will take a different approach, namely to generalize our notion and algorithms for locally decodable codes to locally *list*-decodable codes. Nevertheless, the study of hardness amplification is still of great interest, because it (or variants) can be employed in settings where doing a global encoding of the function is infeasible (e.g. for amplifying the average-case hardness of functions in complexity classes lower than **E**, such as **NP**, and for amplifying the security of cryptographic primitives). We remark that results on hardness amplification can be interpreted in a coding-theoretic language as well, as converting locally decodable codes with a small decoding distance into locally list-decodable codes with a large decoding distance.

7.6.2 Definition

We would like to formulate a notion of local *list*-decoding to enable us to have binary codes that are locally decodable from distances close to $1/2$. This is slightly trickier to define, since for any

function g, there may be several codewords $\hat{f}_1, \hat{f}_2, ..., \hat{f}_s$ that are close to g. So what should our decoding algorithm do? One option would be for the decoding algorithm, on input x, to output a set of values $\mathrm{Dec}^g(x) \subset \Sigma$ that is guaranteed to contain $\hat{f}_1(x), \hat{f_2}(x), ...\hat{f}_s(x)$ with high probability. However, this is not very useful, e.g the list could always be $\mathrm{Dec}^g(x) = \Sigma$. However, rather than outputting each of these values, we want to be able to specify to our decoder *which* $\hat{f}_i(x)$ to output. We do this with a two-phase decoding algorithm. The probabilistic algorithms that accomplish these phases will be referred to as Dec_1 and Dec_2:

(1) Dec_1, using g as an oracle, returns a list of advice strings $a_1, a_2, ..., a_2$, which can be thought of as "labels" for each of the codewords close to g.
(2) Dec_2 (again, using oracle access to g), takes input x and a_i, and outputs $\hat{f}_i(x)$.

The picture for Dec_2 is much like our old decoder, but it takes an extra input a_i corresponding to one of the outputs of Dec_1:

More formally:

Definition 7.51. A *local δ list-decoding algorithm* for a code Enc is a pair of probabilistic oracle algorithms $(\mathrm{Dec}_1, \mathrm{Dec}_2)$ such that for all received words g and all codewords $\hat{f} = \mathrm{Enc}(f)$ with $\Delta(\hat{f}, g) < \delta$, the following holds. With probability at least $1/2$ over $(a_1, ..., a_s) \leftarrow \mathrm{Dec}_1^g$, there exists an $i \in [s]$ such that

$$\forall x, \Pr[\mathrm{Dec}_2^g(x, a_i) = f(x)] \geq 2/3.$$

To help clarify this definition, we make the following remarks. First, we don't require that for all $j, \mathrm{Dec}_2^g(x, a_j)$ are codewords, or even that they're close to s; in other words some of the a_j's may be junk. Second, we don't explicitly require a bound on the list size s, but certainly it cannot be larger than the running time of Dec_1.

As we did for locally (unique-)decodable codes, we can define a *local δ list-correcting algorithm*, where Dec^2 should recover arbitrary symbols of the codeword $\hat{f}$ rather than the message f. Analogously to Lemma 7.36, this implies the above definition if the code is systematic.

Proposition 7.33 shows how locally decodable codes converts functions that are hard in the worst case to ones that are hard on average. The same is true for local list-decoding:

Proposition 7.52. Let Enc be an error-correcting code with local δ-list-decoding algorithm $(\mathrm{Dec}_1, \mathrm{Dec}_2)$ where Dec_2 runs in time at most t_{Dec}, and let f be worst-case hard for non-uniform time t. Then $\hat{f} = \mathrm{Enc}(f)$ is (t', δ) average-case hard, where $t' = t/t_{\mathrm{Dec}}$.

Proof. Suppose for contradiction that $\hat{f}$ is not (t', δ)-hard. Then some algorithm A running in time t' computes $\hat{f}$ with error probability smaller than δ. But if Enc has a local δ list-decoding algorithm, then (with A playing the role of g) that means there exists a_i (one of the possible outputs of Dec_1^A), such that $\mathrm{Dec}_2^A(\cdot, a_i)$ computes $f(\cdot)$ everywhere. Hardwiring a_i as advice, $\mathrm{Dec}_2^A(\cdot, a_i)$ is a nonuniform algorithm running in time at most $\mathrm{time}(A) \cdot \mathrm{time}(\mathrm{Dec}_2) \leq t$. □

Note that, in contrast to Proposition 7.33, here we are using nonuniformity more crucially, in order to select the right function from the list of possible decodings.

7.6.3 Local List-Decoding Reed–Muller Codes

Theorem 7.53. There is a universal constant c such that the q-ary Reed–Muller code of degree d and dimension m over can be locally $(1-\varepsilon)$-list-corrected in time $\text{poly}(q,m)$ for $\varepsilon = c\sqrt{d/q}$.

Note that the distance at which list-decoding can be done approaches 1 as $q/d \to \infty$. It matches the bound for list-decoding Reed–Solomon codes (Theorem **??**) up to the constant c. However, as m increases, the running time of the decoder ($\text{poly}(q,m)$) becomes much smaller than the block length ($q^m \cdot \log q$), at the price of a reduced rate ($\binom{m+d}{m}/q^m$).

Proof. Suppose we are given an oracle $g : \mathbb{F}^m \to \mathbb{F}$ that is $(1-\varepsilon)$ close to some unkown polynomial $p : \mathbb{F}^m \to \mathbb{F}$, and that we are given an $x \in \mathbb{F}^m$. Our goal is is describe two algorithms, Dec_1 and Dec_2, where Dec_2 is able to compute $p(x)$ using a piece of Dec_1's output (i.e. advice).

The advice that we will give to Dec_2 is the value of p on a single point. Dec_1 can easily generate a (reasonably small) list that contains one such point by choosing a random $y \in \mathbb{F}^m$, and outputting all pairs (y,z), for $z \in \mathbb{F}$. More formally:

Algorithm 7.54 (Reed–Muller Local List-Decoder Dec_1).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d

(1) Choose $y \overset{\text{R}}{\leftarrow} \mathbb{F}^m$
(2) Output $\{(y,z) \ : \ z \in \mathbb{F}\}$

Now, the task of Dec_2 is to calculate $p(x)$, given the value of p on some point y. Dec_2 does this by looking at g restricted to the line through x and y, and using the RS list-decoding algorithm to find the univariate polynomials $q_1, q_2, ..., q_t$ that are close to g. If exactly one of these polynomials q_i agrees with p on the test point y, then we can be reasonably confident that $q_i(x) = p(x)$. Specifically:

Algorithm 7.55 (Reed–Muller Local List-Corrector Dec_2).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, advice $(y,z) \in \mathbb{F}^m \times \mathbb{F}$, and a degree parameter d

(1) Let $\ell = \ell_{x,y-x} : \mathbb{F} \to \mathbb{F}^m$ be the line through x and y (so that $\ell(0) = x$ and $\ell(1) = y$).
(2) Run the $(1-\varepsilon/2)$-list-decoder for Reed–Solomon Codes (Theorem 5.19) on $g|_\ell$ to get all univariate polys $q_1 ... q_t$ that agree with $g|_\ell$ in greater than an $\varepsilon/2$ fraction of points.
(3) If there exists a unique i such that $q_i(1) = z$, output $q_i(0)$. Otherwise, fail.

Now that we have fully specified the algorithms, it remains to analyze them and show that they work with the desired probabilities. Observe that it suffices to compute p on at $> 11/12$ of the points x, because then we can apply the unique local decoding algorithm from last time. Therefore, to finish the proof of the theorem we must prove the following lemma

Claim 7.56. Suppose that $g : \mathbb{F}^m \to \mathbb{F}$ has agreement greater than ε with a polynomial p (i.e. g has distance less than $1-\varepsilon$ from p) of degree at most d. For at least $1/2$ of the points $y \in \mathbb{F}^m$ the following holds for greater than an $11/12$ fraction of lines ℓ going through y:

(1) $\mathrm{agr}(g|_\ell, p|_\ell) > \varepsilon/2$.
(2) There does not exist any univariate polynomial q of degree at most d other than $p|_\ell$ such that $\mathrm{agr}(g|_\ell, q) > \varepsilon/2$ and $q(y) = p(y)$.

Proof of claim: It suffices to show that Items 1 and 2 hold with probability 0.99 over random y, ℓ; then we can apply Markov's inequality to finish the job.

Item 1 holds by pairwise independence. If the line ℓ is chosen randomly, then the q points on ℓ are pairwise independent samples of $\mathbb{F}^m$. Note that the expected agreement between $g|_\ell$ and $p|_\ell$ is simply the agreement between $g|_\ell$ and $p|_\ell$, which is greater than ε by hypothesis. So by the Pairwise-Independent Tail Inequality (Prop. 3.27), $\Pr[\mathrm{agr}(g|_\ell, p|_\ell) \le \varepsilon/2] < (1/q \cdot (\varepsilon/2)^2)$, which can be made < 0.01 for a large enough choice of the constant c in $\varepsilon = c\sqrt{d/q}$.

To prove Item 2, we imagine first choosing the line ℓ uniformly at random from all lines in $\mathbb{F}^m$, and then choosing y uniformly at random from the points on ℓ (reparametrizing ℓ so that $\ell(1) = y$). Once we choose ℓ, we can let $q_1, ...q_t$ be all polynomials of degree at most d, other than $p|_\ell$, that have agreement greater than $\varepsilon/2$ with $g|_\ell$. (Note that this list is independent of the parametrization of ℓ, i.e. if $\ell'(t) = \ell(at+b)$ for $a \neq 0$ then $p|_{\ell'}$ and $q_i'(t) = q_i(at+b)$ have agreement equal to $\mathrm{agr}(p|_\ell, q_i)$.) By the list-decodability of Reed–Solomon Codes (Proposition 5.15), we have $t = O(\sqrt{q/d})$.

Now, since two distinct polynomials can agree in at most d points, when we choose a random point $y \xleftarrow{R} \ell$, the probability that q_i and p agree at y is at most d/q. After reparameterization of ℓ so that $\ell(1) = y$, this gives

$$\Pr_y[\exists i \ : \ q_i(1) = p(1)] \le t \cdot \frac{d}{q} = O\left(\sqrt{\frac{d}{q}}\right).$$

This can also be made < 0.01 for large enough choice of the constant c (since we may assume $q/d > c^2$, else $\varepsilon = 1$ and the result is trivial). □

□

7.6.4 Putting it Together

To obtain a locally list-*decodable* (rather than list-*correctible*) code, we again use the low-degree extension (Lemma 7.45) to obtain a systematic encoding. As before, to encode messages of length $\ell = \log L$, we apply Lemma 7.45 with $|\mathbb{H}| = \lceil \sqrt{q} \rceil$ and $m = \lceil \ell / \log |\mathbb{H}|$, for total degree $d \le \sqrt{q} \cdot \ell$. To decode from a $1-\varepsilon$ fraction of errors using Theorem 7.53, we need $c\sqrt{d/q} \le \varepsilon$, which follows if $q \ge c^2\ell^2/\varepsilon^4$. This yields the following locally list-decodable codes:

Theorem 7.57. For every $L \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$, with blocklength $\hat{L} = \mathrm{poly}(L, 1/\varepsilon)$ and alphabet size $|\Sigma| = \mathrm{poly}(\log L, 1/\varepsilon)$, that has a local $(1-\varepsilon)$-list-decoder running in time $\mathrm{poly}(\log L, 1/\varepsilon)$.

Concatenating the code with a Hadamard code, similarly to Problem 5.2, we obtain:

Theorem 7.58. For every $L \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\hat{L}}$ with blocklength $\hat{L} = \mathrm{poly}(L, 1/\varepsilon)$ that has a local $(1/2-\varepsilon)$-list-decoder running in time $\mathrm{poly}(\log L, 1/\varepsilon)$.

Using Proposition 7.52, we get the following hardness amplification result:

Theorem 7.59. If there exists $f\colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against non-uniform time $t(\ell)$, then there exists $\hat{f}\colon \{0,1\}^{O(\ell)} \to \{0,1\}$ in $\mathbf{E}$ that is $(1/2 - 1/t'(\ell))$ average-case hard against (non-uniform) time $t'(\ell)$ for $t'(\ell) = t(\ell)^{\Omega(1)}/\mathrm{poly}(\ell)$.

Combining this with Theorem 7.18 and Corollary 7.19, we get:

Theorem 7.60. For $t : \mathbb{N} \to \mathbb{N}$, suppose that there is a function $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(\ell)})$ [6] such that for every input length $\ell \in \mathbb{N}$, f is worst-case hard for nonuniform time $t(\ell)$. Then for every $m \in \mathbb{N}$, there is an $(m, 1/m)$ pseudorandom generator $G : \{0,1\}^{\ell'(m)} \to \{0,1\}^m$ with seed length $\ell'(m) = O(t^{-1}(\mathrm{poly}(m))^2/\log m)$ that is computable in time $2^{O(\ell'(m))}$.

Corollary 7.61. Suppose that $\mathbf{E}$ has function $f : \{0,1\}^\ell \to \{0,1\}$ that is worst-case hard for nonuniform time $t(\ell)$.

(1) If $t(\ell) = 2^{\Omega(\ell)}$, then $\mathbf{BPP} = \mathbf{P}$.
(2) If $t(\ell) = 2^{\ell^{\Omega(1)}}$, then $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$.
(3) If $t(\ell) = \ell^{\omega(1)}$, then $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$.

We note that the hypotheses in these results are simply asserting that there are problems in $\mathbf{E}$ of high circuit complexity, which is quite plausible. Indeed, many common $\mathbf{NP}$-complete problems, such as SAT, are in $\mathbf{E}$ and are commonly believed to have circuit complexity $2^{\Omega(\ell)}$ (though we seem very far from proving it). Thus, we have a "win-win" situation, either we can derandomize all of $\mathbf{BPP}$ *or* SAT has significantly faster (nonuniform) algorithms than currently known.

Problem **??** establishes a converse to Theorem 7.60, showing that pseudorandom generators imply circuit lower bounds. The equivalence is fairly tight, except for the fact that Theorem 7.60 has seed length $\ell'(m) = O(t^{-1}(\mathrm{poly}(m))^2/\log m)$ instead of $\ell'(m) = O(t^{-1}(\mathrm{poly}(m)))$, and actually it is known how to close this gap too (via a different construction, which is more algebraic and constructs PRGs directly from worst-case hard functions).

[6] $\mathbf{E}$ should be contrasted with the larger class $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(\ell)})$

For Corollary 7.61, however, there is only a partial converse known. Specifically it is known that $\mathbf{prBPP} = \mathbf{prP}$ implies superpolynomial circuit lower bounds for *nondeterministic* exponential time ($\mathbf{NE}$ or $\mathbf{NEXP}$). (We remark that Corollary 7.61, as well as most of the derandomization results we've seen, apply equally well to $\mathbf{prBPP}$ as $\mathbf{BPP}$.) On the other hand, from the *uniform* lower bound $\mathbf{EXP} \neq \mathbf{BPP}$, it is known how to get a subexponential-time *average-case* derandomization of $\mathbf{BPP}$ (specifically, the derandomization is correct on *most* inputs, and it is infeasible to find the inputs on which it errs). The following remains open:

Open Problem 7.62. Does $\mathbf{BPP} = \mathbf{P}$ imply superpolynomial circuit lower bounds for $\mathbf{E}$ (equivalently, $\mathbf{EXP}$)?

Technical Comment. Consider Item **??**, which assumes that there is a problem in $\mathbf{E}$ of superpolynomial circuit complexity. This sounds similar to assuming that $\mathbf{E} \not\subset \mathbf{P/poly}$ (which happens to be equivalent to $\mathbf{EXP} \not\subset \mathbf{P/poly}$). However, the latter assumption is a bit weaker, because it only guarantees that there is a function $f \in \mathbf{E}$ and a function $t(\ell) = \ell^{\omega(1)}$ such that f has complexity at least $t(\ell)$ for *infinitely many* ℓ. Theorem 7.60 and Corollary 7.61 assume that f has complexity at least $t(\ell)$ for all (or all but finitely many) ℓ; equivalently f is not in $\mathbf{i.o.P/poly}$, the class of functions that are computable by poly-sized circuits for infinitely many input lengths. We need the stronger assumptions because we want to build a generator $G : \{0,1\}^{\ell'(m)} \to \{0,1\}^m$ that is pseudorandom for all output lengths m, in order to get derandomizations of $\mathbf{BPP}$ algorithms that are correct on all input lengths. However, there are alternate forms of these results, where the "infinitely often" is moved from the hypothesis to the conclusion. For example, if $\mathbf{E} \not\subset \mathbf{P/poly}$, we can conclude that $\mathbf{BPP} \subseteq \mathbf{i.o.-SUBEXP}$, where $\mathbf{i.o.-SUBEXP}$ denotes the class of languages having deterministic subexponential-time algorithms that are correct for infinitely many input lengths. Even though these "infinitely often" issues need to be treated with care for correctness, it would be quite unexpected if the complexity of problems in $\mathbf{E}$ and $\mathbf{BPP}$ oscillated as a function of input length in such a strange way that they made a real difference.

7.6.5 Black-Box Constructions and Implications

Similarly to our discussion after Theorem 7.23, the pseudorandom generator construction in the previous section is very general. The construction shows how to take *any* function $f : \{0,1\}^\ell \to \{0,1\}$ and use it as a subroutine (oracle) to compute a generator $G^f : \{0,1\}^d \to \{0,1\}^m$ whose pseudorandomness can be related to the hardness of f; the only place that we use the fact that $f \in \mathbf{E}$ is to deduce that G^f is computable in $\mathbf{E}$. The proof that G^f is pseudorandom is also very general. We showed how to take any T that distinguishes the output of $G^f(U_d)$ from U_m and use it as a subroutine (oracle) to build an efficient nonuniform algorithm R such that R^T computes f. The only place that we use the fact that T is itself an efficient nonuniform algorithm is to deduce that R^T is an efficient nonuniform algorithm, contradicting the worst-case hardness of f.

7.7 Exercises

To be written. (See problem sets from Harvard Course CS 225 "Pseudorandomness" at `http://seas.harvard.edu/~salil/cs225`.)

7.8 Chapter Notes and References

To be written.

8

Conclusions

8.1 A Unified Theory of Pseudorandomness

To be written. See survey article "The Unified Theory of Pseudorandomness" at `http://seas.harvard.edu/~salil/research/unified-abs.html`.

8.2 Other Topics in Pseudorandomness

To be written.

References

[Adl] L. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978)*, pages 75–83. IEEE, Long Beach, Calif., 1978.

[AB] M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. *Journal of the ACM*, 50(4):429–443 (electronic), 2003.

[AKS1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics. Second Series*, 160(2):781–793, 2004.

[AKS2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[AKS3] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic Simulation in LOGSPACE. In *19th Annual ACM Symposium on Theory of Computing*, pages 132–140, New York City, 25–27 May 1987.

[AKL$^+$] 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 Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979)*, pages 218–223. IEEE, New York, 1979.

[Alo1] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986. Theory of computing (Singer Island, Fla., 1984).

[Alo2] N. Alon. Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory. *Combinatorica*, 6(3):207–219, 1986.

[AC1] N. Alon and M. R. Capalbo. Explicit Unique-Neighbor Expanders. In *43rd Symposium on Foundations of Computer Science (Vancouver, BC, 2002)*, pages 73–79. IEEE, 2002.

[AC2] N. Alon and F. R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72(1-3):15–19, 1988. Proceedings of the First Japan Conference on Graph Theory and Applications (Hakone, 1986).

[AGKS] N. Alon, V. Guruswami, T. Kaufman, and M. Sudan. Guessing secrets efficiently via list decoding. *ACM Transactions on Algorithms*, 3(4):Art. 42, 16, 2007.

[AMS] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1, part 2):137–147, 1999.

[AM] N. Alon and V. D. Milman. Eigenvalues, Expanders and Superconcentrators (Extended Abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 320–322, Singer Island, Florida, 24–26 Oct. 1984. IEEE.

[AS1] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience [John Wiley & Sons], New York, second edition, 2000. With an appendix on the life and work of Paul Erdős.

[AS2] N. Alon and B. Sudakov. Bipartite subgraphs and the smallest eigenvalue. *Combinatorics, Probability and Computing*, 9(1):1–12, 2000.

[ACRT] A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116 (electronic), 1999.

[Arm] R. Armoni. On the derandomization of space-bounded computations. In *Randomization and approximation techniques in computer science (Barcelona, 1998)*, volume 1518 of *Lecture Notes in Comput. Sci.*, pages 47–59. Springer, Berlin, 1998.

[AB] S. Arora and B. Barak. *Computational complexity.* Cambridge University Press, Cambridge, 2009. A modern approach.

[ALM+] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[AS] S. Arora and S. Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *Journal of the ACM*, 45(1):70–122, Jan. 1998.

[Art] M. Artin. *Algebra.* Prentice Hall Inc., Englewood Cliffs, NJ, 1991.

[BM] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[BRSW] B. Barak, A. Rao, R. Shaltiel, and A. Wigderson. 2-source dispersers for sub-polynomial entropy and Ramsey graphs beating the Frankl-Wilson construction. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 671–680, New York, 2006. ACM.

[Bas] L. A. Bassalygo. Asymptotically optimal switching circuits. *Problems of Information Transmission*, 17(3):206–211, 1981.

[BSS] J. D. Batson, D. A. Spielman, and N. Srivastava. Twice-ramanujan sparsifiers. In *41st Annual ACM Symposium on Theory of Computing (Bethesda, MD)*, pages 255–262. ACM, 2009.

[BGG] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3(4):319–354, 1993.

[BR] M. Bellare and J. Rompel. Randomness-Efficient Oblivious Sampling. In *35th Annual Symposium on Foundations of Computer Science*, pages 276–287, Santa Fe, New Mexico, 20–22 Nov. 1994. IEEE.

[BT] A. Ben-Aroya and A. Ta-Shma. A combinatorial construction of almost-Ramanujan graphs using the zig-zag product. In *40th Annual ACM Symposium on Theory of Computing (Victoria, British Columbia)*, pages 325–334. ACM, 2008.

[BBR] C. H. Bennett, G. Brassard, and J.-M. Robert. Privacy amplification by public discussion. *SIAM Journal on Computing*, 17(2):210–229, 1988. Special issue on cryptography.

[Ber1] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

[Ber2] E. R. Berlekamp. *Algebraic coding theory.* McGraw-Hill Book Co., New York, 1968.

[Ber3] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 1970.

[BL] Y. Bilu and N. Linial. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519, 2006.

[BM] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, 13(4):850–864, Nov. 1984.

[BvzGH] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52(3):241–256, 1982.

[Bro] A. Z. Broder. How hard is to marry at random? (On the approximation of the permanent). In *18th Annual ACM Symposium on Theory of Computing (Berkeley, CA),*, pages 50–58. ACM, 1986.

[BF] H. Buhrman and L. Fortnow. One-sided two-sided error in probabilistic computation. In *STACS 99 (Trier)*, volume 1563 of *Lecture Notes in Comput. Sci.*, pages 100–109. Springer, Berlin, 1999.

[BMRV] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM Journal on Computing*, 31(6):1723–1744 (electronic), 2002.

[BCS] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[CEG] R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995.

[CRVW] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness Conductors and Constant-Degree Lossless Expanders. In *34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 659–668, Montréal, CA, May 2002. ACM. Joint session with *CCC '02*.

[CW] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[Che1] J. Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. In *Problems in analysis (Papers dedicated to Salomon Bochner, 1969)*, pages 195–199. Princeton Univ. Press, Princeton, N. J., 1970.

[Che2] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[CG1] B. Chor and O. Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, 1989.

[CG2] F. Chung and R. Graham. Sparse quasi-random graphs. *Combinatorica*, 22(2):217–244, 2002. Special issue: Paul Erdős and his mathematics.

[CG3] F. Chung and R. Graham. Quasi-random graphs with given degree sequences. *Random Structures & Algorithms*, 32(1):1–19, 2008.

[CGL] F. Chung, R. Graham, and T. Leighton. Guessing secrets. *Electronic Journal of Combinatorics*, 8(1):Research Paper 13, 25 pp. (electronic), 2001.

[Chu] F. R. K. Chung. Diameters and eigenvalues. *Journal of the American Mathematical Society*, 2(2):187–196, 1989.

[CGW] F. R. K. Chung, R. L. Graham, and R. M. Wilson. Quasi-random graphs. *Combinatorica*, 9(4):345–362, 1989.

[CW1] A. Cohen and A. Wigderson. Dispersers, Deterministic Amplification, and Weak Random Sources (Extended Abstract). In *30th Annual Symposium on Foundations of Computer Science (Research Triangle Park, North Carolina)*, pages 14–19. IEEE, 1989.

[CW2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[Csa] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5(4):618–623, 1976.

[DSV] G. Davidoff, P. Sarnak, and A. Valette. *Elementary number theory, group theory, and Ramanujan graphs*, volume 55 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2003.

[DL] R. A. DeMillo and R. J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978.

[Din] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):article 12, 44 pages (electronic), 2007.

[DP] D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[DS] Z. Dvir and A. Shpilka. Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. *SIAM Journal on Computing*, 36(5):1404–1434 (electronic), 2006/07.

[Eli1] P. Elias. *List decoding for noisy channels*. Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Mass., Rep. No. 335, 1957.

[Eli2] P. Elias. The Efficient Construction of an Unbiased Random Sequence. *The Annals of Mathematical Statistics*, 43(3):865–870, June 1972.

[Eli3] P. Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37(1):5–12, 1991.

[Erd] P. Erdős. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53:292–294, 1947.

[EFF] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51(1-2):79–89, 1985.

[ESY] S. Even, A. L. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[FGL$^+$] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.

[Fil] J. A. Fill. Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the exclusion process. *Annals of Applied Probability*, 1(1):62–87, 1991.

[For] G. D. Forney. *Concatenated Codes*. MIT Press, 1966.

[FKS] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[Fri] J. Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *Memoirs of the American Mathematical Society*, 195(910):viii+100, 2008.

[GG] O. Gabber and Z. Galil. Explicit Constructions of Linear-Sized Superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, June 1981.

[Gal] R. G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, 1963.

[Gil1] E. Gilbert. A comparison of signalling alphabets. *Bell Systems Technical Journal*, 31:504–522, 1952.

[Gil2] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[Gil3] D. Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220 (electronic), 1998.

[GW] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[Gol1] O. Goldreich. A Sample of Samplers - A Computational Perspective on Sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(20), 1997.

[Gol2] O. Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1999.

[Gol3] O. Goldreich. *Foundations of cryptography*. Cambridge University Press, Cambridge, 2001. Basic tools.

[Gol4] O. Goldreich. *Foundations of cryptography. II*. Cambridge University Press, Cambridge, 2004. Basic Applications.

[Gol5] O. Goldreich. On promise problems: a survey. In *Theoretical computer science*, volume 3895 of *Lecture Notes in Comput. Sci.*, pages 254–290. Springer, Berlin, 2006.

[Gol6] O. Goldreich. *Computational complexity: a conceptual perspective*. Cambridge University Press, Cambridge, 2008.

[Gol7] O. Goldreich. Pseudorandom Generators: A Primer. `http://www.wisdom.weizmann.ac.il/ oded/prg-primer.html`, July 2008. Revised version of [Gol6, Ch. 8].

[GL] O. Goldreich and L. A. Levin. A Hard-Core Predicate for all One-Way Functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[GMW] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or All languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

[GRS] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning Polynomials with Queries — the Highly Noisy Case. Technical Report TR98-060, Electronic Colloquium on Computational Complexity, 1998. Preliminary version in FOCS '95.

[GW] O. Goldreich and A. Wigderson. Tiny Families of Functions with Random Properties: A Quality-Size Trade-off for Hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

[GM] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gur1] V. Guruswami. Guest column: error-correcting codes and expander graphs. *SIGACT News*, 35(3):25–41, 2004.

[Gur2] V. Guruswami. *Algorithmic Results in List Decoding*, volume 2, number 2 of *Foundations and Trends in Theoretical Computer Science*. now publishers, 2006.

[GHK] V. Guruswami, J. Hastad, and S. Kopparty. On the List-Decodability of Random Linear Codes. `arXiv:1001.1386v1`, January 2010.

[GHSZ] V. Guruswami, J. Håstad, M. Sudan, and D. Zuckerman. Combinatorial bounds for list decoding. *IEEE Transactions on Information Theory*, 48(5):1021–1034, 2002.

[GR] V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[GS1] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *Institute of Electrical and Electronics Engineers. Transactions on Information Theory*, 45(6):1757–1767, 1999.

[GS2] V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. In *STOC*, pages 181–190, 2000.

[GS3] V. Guruswami and M. Sudan. Extensions to the Johnson Bound. Unpublished Manuscript, February 2001.

[GUV] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced Expanders and Randomness Extractors from Parvaresh–Vardy Codes. *Journal of the ACM*, 56(4):1–34, 2009.

[Ham] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29:147–160, 1950.

[HS] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[Har] N. J. A. Harvey. Algebraic Structures and Algorithms for Matching and Matroid Problems. In *47th Annual IEEE Symposium on Foundations of Computer Science (Berkeley, CA)*, pages 531–542. IEEE, 2006.

[HILL] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396 (electronic), 1999.

[Hea] A. D. Healy. Randomness-efficient sampling within NC^1. *Computational Complexity*, 17(1):3–37, 2008.

[Hoe] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hof] A. J. Hoffman. On eigenvalues and colorings of graphs. In *Graph Theory and its Applications (Proc. Advanced Sem., Math. Research Center, Univ. of Wisconsin, Madison, Wis., 1969)*, pages 79–91. Academic Press, New York, 1970.

[HLW] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the AMS*, 43(4):439–561, 2006.

[IZ] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *30th Annual Symposium on Foundations of Computer Science (Research Triangle Park, North Carolina)*, pages 248–253. IEEE, 1989.

[IM] K. Iwama and H. Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In *Mathematical foundations of computer science 2002*, volume 2420 of *Lecture Notes in Comput. Sci.*, pages 353–364. Springer, Berlin, 2002.

[JS] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

[JSV] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697 (electronic), 2004.

[JM] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987.

[Jof1] A. Joffe. On a sequence of almost deterministic pairwise independent random variables. *Proceedings of the American Mathematical Society*, 29:381–382, 1971.

[Jof2] A. Joffe. On a set of almost deterministic k-independent random variables. *Annals of Probability*, 2(1):161–162, 1974.

[Joh] S. M. Johnson. A new upper bound for error-correcting codes. *IRE Transactions on Information Theory*, IT-8:203–207, 1962.

[Kah] N. Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, 1995.

[KLNS] J. D. Kahn, N. Linial, N. Nisan, and M. E. Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, 1989.

[KPS] R. Karp, N. Pippenger, and M. Sipser. A time-randomness tradeoff. In *AMS Conference on Probabilistic Computational Complexity*, Durham, New Hampshire, 1985.

[KL] R. M. Karp and R. J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique. Revue Internationale. IIe Série*, 28(3-4):191–209, 1982.

[KLM] R. M. Karp, M. Luby, and N. Madras. Monte Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[KUW] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in Random NC. *Combinatorica*, 6(1):35–48, 1986.

[KL] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC Cryptography and Network Security. Chapman & Hall/CRC, Boca Raton, FL, 2008.

[KS] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

[LR] O. Lachish and R. Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In *33rd Annual ACM Symposium on Theory of Computing*, pages 399–408 (electronic), New York, 2001. ACM.

[Lan] H. O. Lancaster. Pairwise statistical independence. *Annals of Mathematical Statistics*, 36:1313–1317, 1965.

[Lau] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[Lei] F. T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, and hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[LV] D. Lewin and S. Vadhan. Checking polynomial identities over any field: towards a derandomization? In *30th Annual ACM Symposium on the Theory of Computing (Dallas, TX)*, pages 438–447. ACM, New York, 1999.

[LN] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, Cambridge, first edition, 1994.

[Lov1] L. Lovász. On determinants, matchings, and random algorithms. In *Fundamentals of Computation Theory (Berlin/Wendisch-Rietz)*, pages 565–574, 1979.

[Lov2] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.

[Lov3] L. Lovász. *Combinatorial problems and exercises*. AMS Chelsea Publishing, Providence, RI, second edition, 2007.

[Lub] A. Lubotzky. *Discrete groups, expanding graphs and invariant measures*, volume 125 of *Progress in Mathematics*. Birkhäuser Verlag, Basel, 1994. With an appendix by Jonathan D. Rogawski.

[LPS] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[Lub1] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[Lub2] M. Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[LW] M. Luby and A. Wigderson. *Pairwise Independence and Derandomization*, volume 1, number 4 of *Foundations and Trends in Theoretical Computer Science*. now publishers, 2005.

[MS1] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.

[MS2] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes. I*. North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.

[MS3] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes. II*. North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.

[Mar1] G. A. Margulis. Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80, 1973.

[Mar2] G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredači Informacii*, 24(1):51–60, 1988.

[MR] R. Martin and D. Randall. Disjoint decomposition of Markov chains and sampling circuits in Cayley graphs. *Combinatorics, Probability and Computing*, 15(3):411–448, 2006.

[Mih] M. Mihail. Conductance and Convergence of Markov Chains-A Combinatorial Treatment of Expanders. In *30th Annual Symposium on Foundations of Computer Science (Research Triangle Park, North Carolina)*, pages 526–531. IEEE, 1989.

[Mil1] G. L. Miller. Riemann's Hypothesis and Tests for Primality. *Journal of Computer and System Sciences*, 13(3):300–317, Dec. 1976.

[Mil2] P. Miltersen. *Handbook of Randomized Computing*, chapter Derandomizing Complexity Classes. Kluwer, 2001.

[MU] M. Mitzenmacher and E. Upfal. *Probability and computing*. Cambridge University Press, Cambridge, 2005. Randomized algorithms and probabilistic analysis.

[MR] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[MS] M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *45th Symposium on Foundations of Computer Science (Rome, Italy)*, pages 248–255. IEEE Computer Society, 2004.

[Mul] D. E. Muller. Boolean algebras in electric circuit design. *The American Mathematical Monthly*, 61(7, part II):27–28, 1954. Proceedings of the symposium on special topics in applied mathematics, Northwestern University (1953).

[MVV] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[Mut] S. Muthukrishnan. *Data Streams: Algorithms and Applications*, volume 1, number 2 of *Foundations and Trends in Theoretical Computer Science*. now publishers, 2005.

[Nil] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91(2):207–210, 1991.

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

[NT] N. Nisan and A. Ta-Shma. Extracting Randomness: A Survey and New Constructions. *Journal of Computer and System Sciences*, 58(1):148–173, February 1999.

[NW] N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Sciences*, 49(2):149–167, Oct. 1994.

[NZ] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Sciences*, 52(1):43–52, Feb. 1996.

[PV] F. Parvaresh and A. Vardy. Correcting Errors Beyond the Guruswami-Sudan Radius in Polynomial Time. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 285–294. IEEE Computer Society, 2005.

[Per] Y. Peres. Iterating von Neumann's procedure for extracting random bits. *The Annals of Statistics*, 20(1):590–597, 1992.

[Pet] W. W. Peterson. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IRE Transactions on Information Theory*, IT-6:459–470, 1960.

[Pin] M. Pinsker. On the Complexity of a Concentrator. In *7th Annual Teletraffic Conference*, pages 318/1–318/4, Stockholm, 1973.

[Pip] N. Pippenger. On Simultaneous Resource Bounds (Preliminary Version). In *20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico)*, pages 307–311. IEEE, 1979.

[PHB] V. S. Pless, W. C. Huffman, and R. A. Brualdi, editors. *Handbook of coding theory. Vol. I, II.* North-Holland, Amsterdam, 1998.

[Rab] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[Rag] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988. Twenty-Seventh Annual IEEE Symposium on the Foundations of Computer Science (Toronto, ON, 1986).

[Ran] D. Randall. Mixing. In *44th Symposium on Foundations of Computer Science (Cambridge, MA)*, pages 4–15. IEEE Computer Society, 2003.

[Ree] I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IRE Transactions on Information Theory*, PGIT-4:38–49, 1954.

[RS] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8:300–304, 1960.

[Rei] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):Art. 17, 24, 2008.

[RTV] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom Walks in Regular Digraphs and the RL vs. L Problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 457–466, 21–23 May 2006. Preliminary version as *ECCC* TR05-22, February 2005.

[RVW] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders. *Annals of Mathematics*, 155(1), January 2001.

[Ron] D. Ron. Property testing. In *Handbook of randomized computing, Vol. I, II*, volume 9 of *Comb. Optim.*, pages 597–649. Kluwer Acad. Publ., Dordrecht, 2001.

[RV] E. Rozenman and S. Vadhan. Derandomized Squaring of Graphs. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM '05)*, number 3624 in Lecture Notes in Computer Science, pages 436–447, Berkeley, CA, August 2005. Springer.

[Rub] R. Rubinfeld. Sublinear time algorithms. In *International Congress of Mathematicians. Vol. III*, pages 1095–1110. Eur. Math. Soc., Zürich, 2006.

[SG] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[SSZ] M. Saks, A. Srinivasan, and S. Zhou. Explicit OR-dispersers with polylogarithmic degree. *Journal of the ACM*, 45(1):123–154, 1998.

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

[SV] M. Sántha and U. V. Vazirani. Generating quasirandom sequences from semirandom sources. *Journal of Computer and System Sciences*, 33(1):75–87, 1986. Twenty-fifth annual symposium on foundations of computer science (Singer Island, Fla., 1984).

[Sar] P. Sarnak. *Some applications of modular forms*, volume 99 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 1990.

[Sav] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[SSS] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

[Sch] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[Sha1] R. Shaltiel. Recent Developments in Extractors. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, volume 1: Algorithms and Complexity. World Scientific, 2004.

[Sha2] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Sip1] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th Annual ACM Symposium on Theory of Computing*, pages 330–335, Boston, Massachusetts, 25–27 Apr. 1983.

[Sip2] M. Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, 1988. Structure in Complexity Theory Conference (Berkeley, CA, 1986).

[Sip3] M. Sipser. *Introduction to the Theory of Computation.* Course Technology, 2nd edition, 2005.

[SS1] M. Sipser and D. A. Spielman. Expander codes. *IEEE Trans. Inform. Theory*, 42(6, part 1):1710–1722, 1996. Codes and complexity.

[SS2] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[Spe] J. Spencer. *Ten lectures on the probabilistic method*, volume 64 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 1994.

[Spi] D. A. Spielman. Spectral Graph Theory and its Applications. In *48th Symposium on Foundations of Computer Science (FOCS 2007), 21-23 October 2007, Providence, RI, USA, Proceedings*, pages 29–38, 2007.

[Str] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[Sud1] M. Sudan. Decoding of Reed Solomon Codes beyond the Error-Correction Bound. *Journal of Complexity*, 13(1):180–193, Mar. 1997.

[Sud2] M. Sudan. Algorithmic Introduction to Coding Theory. Lecture notes, 2001. `http://people.csail.mit.edu/madhu/FT01/`.

[Sud3] M. Sudan. Essential Coding Theory (Lecture Notes). `http://people.csail.mit.edu/madhu/FT04/`, 2004.

[TZ] A. Ta-Shma and D. Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004.

[Tan] M. R. Tanner. Explicit Concentrators from Generalized N-gons. *SIAM Journal on Algebraic Discrete Methods*, 5(3):287–293, 1984.

[Tre] L. Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879 (electronic), 2001.

[Vad] S. Vadhan. Probabilistic Proof Systems, Part I — Interactive & Zero-Knowledge Proofs. In S. Rudich and A. Wigderson, editors, *Computational Complexity Theory*, volume 10 of *IAS/Park City Mathematics Series*, pages 315–348. American Mathematical Society, 2004.

[Val] L. G. Valiant. Graph-theoretic properties in computational complexity. *Journal of Computer and System Sciences*, 13(3):278–285, 1976.

[Var] R. Varshamov. Estimate of the number of signals in error correcting codes. *Doklady Akad. Nauk SSSR*, 117:739–741, 1957.

[Vaz1] U. V. Vazirani. Towards a Strong Communication Complexity Theory or Generating Quasi-Random Sequences from Two Communicating Slightly-random Sources (Extended Abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 366–378, Providence, Rhode Island, 6–8 May 1985.

[Vaz2] U. V. Vazirani. Efficiency Considerations in Using Semi-random Sources (Extended Abstract). In *STOC*, pages 160–168. ACM, 1987.

[von] J. von Neumann. Various techniques used in conjunction with random digits. In *Collected works. Vol. V: Design of computers, theory of automata and numerical analysis*. The Macmillan Co., New York, 1963.

[WC] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[Woz] J. Wozencraft. List decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.

[Yao] A. C. Yao. Theory and Applications of Trapdoor Functions (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 Nov. 1982. IEEE.

[Zip] R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

[Zuc1] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, 16(4/5):367–391, Oct./Nov. 1996.

[Zuc2] D. Zuckerman. Randomness-Optimal Oblivious Sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.

[ZP] V. V. Zyablov and M. S. Pinsker. List cascade decoding (in Russian). *Problems of Information Transmission*, 17(4):29–33, 1981.