Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–85

DOI: xxxxxx

now
the essence of knowledge

Pseudorandomness I

Salil P. Vadhan[1]

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

Abstract

This is the first volume of a 2-part survey on *pseudorandomness,* the theory of efficiently generating objects that "look random" despite being constructed using little or no randomness. The survey 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

Acknowledgments

My exploration of pseudorandomness began in my graduate and postdoctoral years at MIT and IAS, under the wonderful guidance of Shafi Goldwasser, Oded Goldreich, 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 Shafi, Oded, 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, Ronen Gradwohl, Dan Gutfreund, Venkat Guruswami, Alex Healy, Jesse Kamp, Danny Lewin, Chi-Jen Lu, Michael Mitzenmacher, Shien Jin Ong, Michael Rabin, Anup Rao, Ran Raz, Leo Reyzin, Eyal Rozenman, Madhur Tulsiani, Chris Umans, Emanuele Viola, and David Zuckerman. Needless to say, this list omits many other researchers in the field with whom I have had stimulating discussions on related topics.

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 for his extensive feedback on the lecture notes. Helpful comments and corrections have also been given by Trevor Bass, Zhou Fan, Alex Healy, Andrei Jorza, Shira Mitchell, Jelani Nelson, Yakir Reshef, Shrenik Shah, Michael von Korff, 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. 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.

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 properties. 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 $2\log n$. 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 (e.g. leader election), circuit lower bounds 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 (PRGs): A PRG 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 those of 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 PRG 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 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—producing extractors that are only "mildly" probabilistic.

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, one might say that 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), which are networks 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 proved much harder to find. We will see some explicit constructions in this survey, but they do not always match the bounds given by the probabilistic method (in terms of the degree/expansion tradeoff).

Error-Correcting Codes: Error-correcting codes (ECCs) 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. PRGs, 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 [Sip2]. 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 the 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

All logarithms are base 2 unless otherwise specified. 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.

Throughout, we consider random variables that can take values in arbitrary discrete sets (not just 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 $x \stackrel{R}{\leftarrow} X$ to indicate that x is sampled according to X. For a set S, we write $x \stackrel{R}{\leftarrow} S$ 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. $\Pr[X = X] = 1$. For an event E, we write $X|_E$ to denote the random variable X conditioned on the event E.

1.4 Chapter Notes and References

General introductions to the theory of pseudorandomness (other than this survey) include [Gol2, Mil2].

Recommended textbooks focused on randomized algorithms are [MU, MR]. The first randomized polynomial-time algorithms for PRIMALITY were discovered by Miller and Rabin [Mil1, Rab] and Solovay and Strassen [SS]; a deterministic polynomial-time algorithm was given by Agrawal, Kayal, and Saxena [AKS1]. The first randomized algorithms for approximate counting were found by Karp and Luby [KLM]; the algorithm for counting perfect matchings is due to Jerrum, Sinclair, and Vigoda [JSV], building on [Bro, JS]. The randomized logspace algorithm for UNDIRECTED S-T CONNECTIVITY was given by Aleliunas et al. [AKL$^+$]; it was derandomized by Reingold [Rei]. The randomized parallel algorithm for deciding PERFECT MATCHING is due to Lovász [Lov1]; the search version is handled in [KUW] (see also [MVV]).

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 [AS]. Erdős used this method to prove the existence of Ramsey graphs in [Erd]. Major recent progress on explicit constructions of Ramsey graphs was obtained by Barak, Rao, Shaltiel, and Wigderson [BRSW] via the theory of randomness extractors.

The modern notion of 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 [MS]. For a modern, CS-oriented treatment, we recommend Sudan's lecture notes [Sud2]. Shannon's paper that gave birth to the field and gave a probabilistic construction of error-correcting codes is [Sha2]. The notion of list decoding was proposed by Elias [Eli] and Wozencraft [Woz], and was reinvigorated in the work of Sudan [Sud1]. Recent progress on list decoding is covered in [Gur].

2

The Power of Randomness

2.1 Polynomial Identity Testing

Before we study the derandomization of randomized algorithms, we will need some algorithms to derandomize. This section introduces one such algorithm. It solves the following computational problem.

Computational Problem 2.1. IDENTITY TESTING: given two multivariate polynomials, $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$, decide whether $p = q$.

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

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

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

  We need to specify what space the coefficients of the polynomials will come from; 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), $\mathbb{Z}_p$ (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); but $\mathbb{F}_q$ 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 $p(x_1, \ldots, x_n) = \sum_{i_1, \ldots, i_n} c_{i_1, \ldots, i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$, 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.

- "$p = q$": 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
  $$p(x) = \prod_{\alpha \in \mathbb{F}} (x - \alpha).$$
  for a finite field $\mathbb{F}$.[1] It is easy to see that $p(\alpha) = 0$ for all $\alpha \in \mathbb{F}$, but $p \neq 0$ as a formal polynomial. For us, *equality refers to equality as formal polynomials.*
- "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 Identity Testing, as we can just compare.
  (2) As an "oracle": a black box that, given any point in the domain, gives the value of the polynomial.
  (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 Identity Testing by expanding the polynomials and grouping terms, but the expanded polynomials may have length exponential in the length of the formula, and thus the algorithm is not efficient.
  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 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 both the 2nd and 3rd formulations above (oracles and arithmetic circuits/formulas). It will be convenient to work with the following equivalent version of the problem.

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

That is, we consider the special case of the original problem where $q = 0$. 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 $p' = p - q$.

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

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

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

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

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

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

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 $p \neq 0$. 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 p. 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. Algorithms whose correctness (or efficiency) only holds for randomly chosen inputs 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 representation of the domain $\mathbb{F}$ enabling us to write down and manipulate field elements (e.g. the prime p in case $\mathbb{F} = \mathbb{Z}_p$). Then, if we are given p as an oracle, we have the ability to evaluate p by definition. If we are given p 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 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 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 Identity Testing, Perfect Matching has 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 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 a polynomial number of processors) using an efficient parallel algorithm for determinant.

So to test for a perfect matching efficiently in parallel, just run the 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 non-bipartite graphs, and enable *finding* perfect matchings in the same parallel or sequential time complexity (where the result for sequential time is quite recent).
- Identity Testing has been also used to obtain a randomized algorithm for Primality, which was derandomized fairly recently (2002) to obtain the celebrated deterministic polynomial-time algorithm for Primality. 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 "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 that is initially filled with random bits. For polynomial-time algorithms, this is equivalent to the 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 can be shown to be equivalent to allowing the algorithm *one-way* access to an infinite tape of random bits.

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* $L \subseteq \{0,1\}^*$. 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 an 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.

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 coin tosses.*

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 in polynomial time (e.g. decide more languages):

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

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

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

Using the IDENTITY TESTING algorithm we saw earlier, we can deduce that 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. 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 requires 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 such that exists a probabilistic parallel algorithm A that runs in polylogarithmic time on polynomially many processors, 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**.

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 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 need to take 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

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

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 (aka [$+\varepsilon$]-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). For an appropriate choice of t, choose $x_1, \ldots, x_t \stackrel{R}{\leftarrow} \{0,1\}^m$, query the oracle to obtain $f(x_1), \ldots, f(x_t)$, and output $(\sum_i f(x_i))/t$.

By the Chernoff Bound (Theorem 2.21), we only need to take $t = O(\log(1/\delta)/\varepsilon^2)$ samples to have additive error at most ε with probability at least $1 - \delta$. Note that for constant ε and δ, the sample size is independent of the size of the population (2^m), and we have running time $\text{poly}(m)$ even for $\varepsilon = 1/\text{poly}(m)$ and $\delta = 2^{-\text{poly}(m)}$.

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. Now define two functions

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

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) [+ε]-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 sublinear (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 [+ε]-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 $\Pi_Y \cup \Pi_N$, 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 result proven 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. $\mathbf{prBPP} = \mathbf{prP} \Rightarrow \mathbf{BPP} = \mathbf{P}$.

Now we can consider the following 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]
Inclusion in **prBPP**: Follows from Algorithm 2.24 and the fact that boolean circuits can be evaluated in polynomial time.

Hardness for **prBPP**: Given any promise problem $\Pi \in \mathbf{prBPP}$, we have a probabilistic polynomial-time algorithm A that decides Π with 2-sided error at most 2^{-n} on inputs of length n.

We can view the output of $A(x;r)$ as a function of its input x and its coin tosses r. Note that if x is of length n, then we may assume that r is of length at most $\text{poly}(n)$ without loss of generality (because an algorithm that runs in time at most t can toss at most t coins). For any n, there is a $\text{poly}(n)$-sized circuit $C(x;r)$ that simulates the computation of A for inputs x of length n and coin tosses r of length $\text{poly}(n)$, and moreover C can be constructed in time $\text{poly}(n)$. (See any text on complexity theory for a proof.) 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 $n' = |(C_x, 1/2^n)| = \text{poly}(n)$ and we take n sufficiently large. □

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 the proof of Proposition 2.25 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 information 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:

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

Unfortunately, this problem is **NP**-hard for general circuits (consider $N = 0$), 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 to 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 $\varepsilon(n) \geq 1/\text{poly}(n)$, $[\times(1+\varepsilon)]$-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 $A' \subseteq B$ satisfying the following properties:

(1) $|A'| = |A|$
(2) $|A'| \geq |B|/\text{poly}(n)$, where $n = |\varphi|$.
(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:

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

Now we verify the desired properties:

(1) Clearly $|A| = |A'|$ since A' only contains pairs (i,α) such that the first satisfying clause in α is the i^{th} one.
(2) Also, the size of A' and B can differ by at most a factor of ℓ by construction since for A' we only look at the first satisfying clause and there can only be $m-1$ more elements in B per assignment α.
(3) It is easy to decide membership in A' in linear time.
(4) $|B| = \sum_{i=1}^{\ell} 2^{m-m_i}$, where m_i is the number of literals in clause i.
(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 $t = O(1/(\varepsilon/\ell)^2)$ to be determined below.
(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, we have $\hat{\mu} \in [|A'|/|B| \pm \varepsilon/\ell]$ with high probability (where we write $[\alpha \pm \beta]$ to denote the interval $[\alpha - \beta, \alpha + \beta]$). Thus, with high probability the output of the algorithm satisfies:

$$\hat{\mu} \cdot |B| \in [|A'| \pm \varepsilon |B|/\ell] \subseteq [|A| \pm \varepsilon |A|].$$

□

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 $2^{\text{polylog}(n)}$).

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 $S, T \subseteq V$, define $\text{cut}(S, T) = \{\{u, v\} \in E : u \in S, v \in T\}$, and $\text{cut}(S) = \text{cut}(S, V \setminus S)$.

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

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 $S \subseteq V$. 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, i.e.

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 $\mathrm{IsPath}(G, u, v, k)$ decides whether there is a path of length at most k from u to v.

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

(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 $\mathrm{IsPath}(G, u, w, \lceil k/2 \rceil)$ and $\mathrm{IsPath}(G, w, v, \lfloor k/2 \rfloor)$ accept for some w.

We can solve S-T Connectivity by running $\mathrm{IsPath}(G, s, t, n)$, 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 up to n^4 times:
    (a) If $v = t$, halt and accept.
    (b) Else let $v \xleftarrow{R} \{w : (v, w) \in E\}$.
(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 if G is a d-regular graph, then a random walk of length $\tilde{O}(d^2 n^3)$ from s will hit t with high probability. Note that this suffices for Theorem 2.45, because make an arbitrary undirected graph 3-regular while preserving s-t connectivity by replacing each vertex v with a cycle of length $\deg(v)$. In fact, the algorithm actually works as described above for general undirected graphs and even directed graphs in which each connected component is Eulerian

(indeg(v) = outdeg(v) for every vertex), but we will not prove it here. But it does not work for arbitrary directed graphs. Indeed, it is not difficult to construct directed graphs in which there is a path from s to t but a random walk from s takes exponential time to hit t (Problem 2.9). □

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

Open Problem 2.47. Does **RL** = **L**? Does **BPL** = **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. We call such 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 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 and d-regular undirected graph G on n vertices, we have $\mathrm{hit}(G) = O(d^2 n^3 \log n)$.

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, we start at a probability distribution π concentrated at vertex s, and are interested in the distribution πM^k we get after taking k steps on the graph. Specifically, we'd like to show that it places nonnegligible mass on vertex t for $k = \mathrm{poly}(n)$. 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 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 $\lambda(G) \in [0,1]$. (For undirected regular graphs, this follows from Problem 2.11.)

The following lemma is immediate from the definition of M.

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 $k \in \mathbb{N}$, we have

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

Thus a smaller value of $\lambda(G)$ (equivalently, a larger value of $\gamma(G)$) means that the random walk mixes more quickly. Specifically, for $k = \ln(n/\varepsilon)/\gamma(G)$, it follows that every entry of πM^k has probability mass at least $1/n - (1 - \gamma(G))^k \geq (1 - \varepsilon)/n$. 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. $\text{hit}(G) = O(n \log n/\gamma(G))$.

Proof. As argued above, a walk of length $k = O(\log n/\gamma(G))$ has a probability of at least $1/2n$ of ending at j. Thus, if we do $O(n)$ such walks, we will hit j with probability at least $1/2$. □

Thus we are left with the task of showing that $\gamma(G) \geq 1/\text{poly}(n)$. This is done in Problem 2.11, using a connection with eigenvalues described in the next section.

2.4.3 Eigenvalues

Recall that $v \in \mathbb{R}^n$ 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 2.53 (Spectral Thm for Symmetric Matrices). If M is a symmetric $n \times n$ real matrix with distinct eigenvalues $\mu_1, \ldots, \mu_k$, then the subspaces $W_i = \{x : x \text{ is an eigenvector of eigenvalue } \mu_i\}$ are orthogonal (i.e. $x \in W_i$, $y \in W_j \Rightarrow x \perp y$ 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

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

In Problem 2.11, it is shown that all of the λ_i's have absolute value at most 1. Notice that if they all have have magnitude strictly smaller than 1, then πM^k 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 2.54. Let G be an undirected graph with random-walk matrix M. Let $1 = \lambda_1 \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$ be the eigenvalues of M. 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.

In Problem 2.11, it is shown that:

Theorem 2.55. If G is a connected, nonbipartite, and regular undirected graph, then $\gamma(G) = \Omega(1/(dn)^2)$.

Combining Theorem 2.55 with Corollary 2.52, we deduce Theorem 2.49. (The nonbipartite assumption in Theorem 2.55 can be achieved by adding a self-loop to each vertex, which only increases the hitting time.) We note that the bounds presented here are not tight.

2.4.4 Markov Chain Monte Carlo

Random walks are a very widely used 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 $p(x_1, \ldots, x_n)$ is a nonzero polynomial of degree d over a a field (or integral domain) $\mathbb{F}$ and $S \subseteq \mathbb{F}$, then~~

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

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 (Identity Testing for integer circuits). In this problem, you will show how to do 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 $M \stackrel{R}{\leftarrow} \{1, \ldots, \log^2 N\}$, then

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

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

Problem 2.5 (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 $p(x)$ 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 p). 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 $p(x)$ and $q(x)$, $p(x) \bmod q(x)$ is the remainder when p is divided by q. (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 Identity Testing problem where a polynomial $p(x_1, \ldots, x_n)$ 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 $p(x)$ be a univariate polynomial of degree $\leq D$ over a field $\mathbb{F}$. Prove that there is a constant c such that if $p(x)$ is nonzero (as a formal polynomial) and $q(x)$ is a randomly selected polynomial of degree at most $d = c \log D$, then the probability that $p(x) \bmod q(x)$ 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)

(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$= \{n : n \text{ prime}\}$ using Part 1 together with the previous problem. (In your analysis, remember that the integers modulo n are a field only when n is prime.)

Problem 2.7 (A Chernoff Bound). Let $X_1, \ldots, X_t$ be independent $[0,1]$-valued random variables, and $X = \sum_{i=1}^{t} X_i$.

(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 following Chernoff Bound: $\Pr[X \geq \mathrm{E}[X] + \varepsilon t] \leq e^{-\varepsilon^2 t/4}$. 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 $p : \mathbb{F}^m \to \mathbb{F}$ of degree d is given as an oracle (ie black box) and the problem is to test whether $p = 0$. 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 (Random Walks on Directed Graphs). Show that for every n, there exists a digraph G with n vertices, outdegree 2, and $\mathrm{hit}(G) = 2^{\Omega(n)}$.

Problem 2.10 (Regular Digraphs and Eigenvalues). Let G be a regular digraph with random-walk matrix M.

(1) Show that $\lambda(G)$ is the square root of the absolute value of the second-largest eigenvalue of the symmetric matrix MM^T.
(2) Describe the graph for which MM^T is the random-walk matrix.

Problem 2.11 (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 (or all-ones vector) 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 $\leq 1 - 1/\text{poly}(n, d)$. To do this, it may help to first show that the second largest eigenvalue of M (not necessarily in absolute value) equals

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

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 $\lambda(G) \leq 1 - 1/\textit{poly}(n, d)$.
(6*) Extra credit: Establish the (tight) bound $1 - \Omega(1/d \cdot D \cdot n)$ in Part 4, where D is the diameter of the graph, and show that a simple graph satisfies $D \leq O(n/d)$. (The $1 - \Omega(1/d \cdot D \cdot n)$ bound also holds for Part 5, but you do not need to prove it here.)

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 the Miller–Rabin [Mil1, Rab] and Solovay–Strassen [SS] algorithms for Primality. The randomized algorithm for Identity Testing was independently discovered by DeMillo and Lipton [DL], Schwartz [Sch], and Zippel [Zip]. A deterministic polynomial-time 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 2.8 is from [LV].

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

The randomized algorithm for Perfect Matching is due to Lovász, who also showed how to extend the algorithm to non-bipartite 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]).

For more on parallel algorithms, we refer to the textbook by Leighton [Lei]. The Identity Testing and Primality algorithms of Problems 2.5 and 2.6 are due to Agrawal and Biswas [AB].

Agrawal, Kayal, and Saxena [AKS1] derandomized the PRIMALITY algorithm to prove that PRIMALITY is in **P**.

The randomized complexity classes **RP**, **BPP**, **ZPP**, and **PP** were formally defined by Gill [Gil1], who conjectured that **BPP** $\neq$ **P** (in fact **ZPP** $\neq$ **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. For some other Chernoff Bounds, see [MR]. Problem 2.3 is due to Rabin (cf. [Gil1]).

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]. 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 $\mathbf{NL} \subseteq \mathbf{L}^2$, where $\mathbf{L}^c$ is the class of languages that can be decided deterministic space $O(\log^2 n)$. The latter formulation (and its generalization $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{DSPACE}(s(n)^2)$) 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, Ran].

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 digraphs, or more generally to digraphs where every vertex has the same indegree as outdegree (i.e. where each connected component is Eulerian). See e.g. [Mih, Fil, RTV]. Problem 2.10 is from [Fil].

The Spectral Theorem (Thm. 2.53) can be found in any standard textbook on linear algebra. Problem 2.11, Part 5 is from [Lov2]. 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. (These are both subclasses of the class **IP** of languages having *interactive proof systems*.) 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. For more about interactive proofs, 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:

- 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 attack 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 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+).

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. $\mathbf{BPP} \subseteq \mathbf{EXP}$.

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 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. Then:

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

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

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 $r(n)$ random bits, then $L \in \mathbf{DTIME}(t(n) \cdot 2^{r(n)})$. In particular, if $t(n) = \text{poly}(n)$ and $r(n) = O(\log n)$, 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 $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$? Is $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$?

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 $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, and 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 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 halting problem, the advice string α_n is simply a bit that tells us whether the n'th Turing machine halts or not.

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

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

3.3 Nondeterminism

Although physically unrealistic, nondeterminism has proved be a very useful resource in the study of computational complexity (e.g. leading to the class **NP**). Thus it is natural 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. $\mathbf{RP} \subseteq \mathbf{NP}$.

Proof. Let $L \in \mathbf{RP}$ and A be a randomized algorithm that decides it. A poly-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 $\mathbf{BPP} = \mathbf{NEXP}$! 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,

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

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 $Z_x \subset \{0,1\}^m$ 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 points in $\{0,1\}^m$ such that no "shift" (or "translation") of Z_x covers all the points. (Notice that this is a $\exists\forall$ statement.) Intuitively, this should be possible because Z_x is an exponentially small fraction of $\{0,1\}^m$. On the other hand if $x \notin L$, then for any m points in $\{0,1\}^m$, we will show that there is a "shift" of Z_x that covers all the points. Intuitively, this should be possible because Z_x covers all but an exponentially small fraction of $\{0,1\}^m$.

Formally, by a "shift of Z_x" we mean a set of the form $Z_x \oplus s = \{r \oplus s : r \in Z_x\}$ for some $s \in \{0,1\}^m$; note that $|Z_x \oplus s| = |Z_x|$. We will show

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \exists r_1, r_2, \ldots, r_m \in \{0,1\}^m \forall s \in \{0,1\}^m \quad \neg \bigwedge_{i=1}^{m} (r_i \in Z_x \oplus s) \\ &\Leftrightarrow \quad \exists r_1, r_2, \ldots, r_m \in \{0,1\}^m \, \forall s \in \{0,1\}^m \quad \neg \bigwedge_{i=1}^{m} \left(A(x; r_i \oplus s) = 0\right); \end{aligned}$$

$$
\begin{aligned}
x \notin L \quad &\Rightarrow \quad \forall r_1, r_2, \ldots, r_m \in \{0,1\}^m \, \exists s \in \{0,1\}^m \quad \bigwedge_{i=1}^{m} (r_i \in Z_x \oplus s) \\
&\Leftrightarrow \quad \forall r_1, r_2, \ldots, r_m \in \{0,1\}^m \, \exists s \in \{0,1\}^m \quad \bigwedge_{i=1}^{m} (A(x; r_i \oplus s) = 0) \, .
\end{aligned}
$$

We prove both parts by the Probabilistic Method.

$x \in L$: Choose $R_1, R_2, \ldots, R_m \stackrel{\text{R}}{\leftarrow} \{0,1\}^m$. Then, for every fixed s, Z_x and hence $Z_x \oplus s$ contains less than a 2^{-n} fraction of points in $\{0,1\}^m$, so:

$$
\begin{aligned}
\forall i \quad & \Pr[R_i \in Z_x \oplus s] < 2^{-n} \\
\Rightarrow \quad & \Pr\left[\bigwedge_i (R_i \in Z_x \oplus s)\right] < 2^{-nm}. \\
\Rightarrow \quad & \Pr\left[\exists s \bigwedge_i (R_i \in Z_x \oplus s)\right] < 2^m \cdot 2^{-nm} < 1.
\end{aligned}
$$

Thus there exist $r_1, \ldots, r_m$ such that $\forall s \, \neg \bigwedge_i (r_i \in Z_x \oplus s)$, as desired.

$x \notin L$: Let $r_1, r_2, \ldots, r_m$ be arbitrary, and choose $S \stackrel{\text{R}}{\leftarrow} \{0,1\}^m$ at random. Now Z_x and hence $Z_x \oplus r_i$ contains more than a $1 - 2^{-n}$ fraction of points, so:

$$
\begin{aligned}
\forall i \quad & \Pr[r_i \notin Z_x \oplus S] = \Pr[S \notin Z_x \oplus r_i] < 2^{-n} \\
\Rightarrow \quad & \Pr\left[\bigvee_i (r_i \notin Z_x \oplus S)\right] < m \cdot 2^{-n} < 1.
\end{aligned}
$$

Thus, for every $r_1, \ldots, r_m$, there exists s such that $\bigwedge_i (r_i \in Z_x \oplus s)$, as desired.

□

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 $1 \le i \le m$ 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 3.4.1.) By averaging, there exists an $r_{i+1} \in \{0,1\}$ such that $P(r_1, r_2, \ldots, r_i, r_{i+1}) \ge$

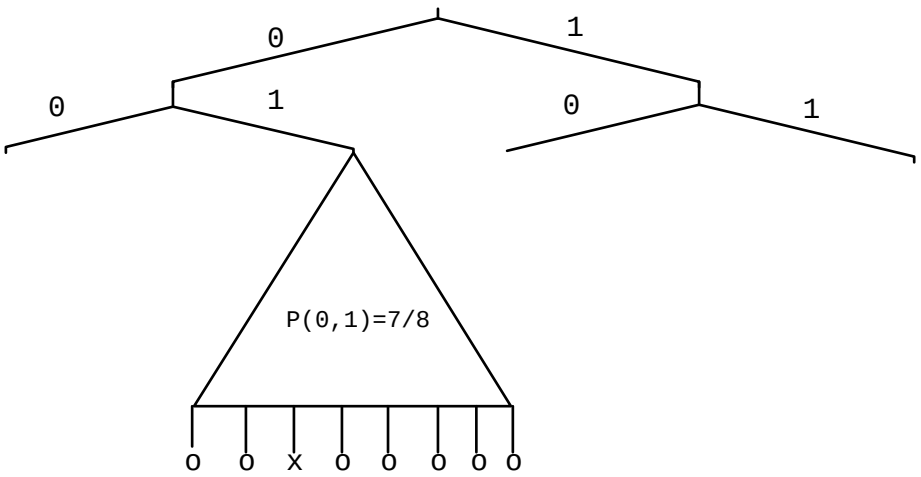

Fig. 3.1 An example of $P(r_1, r_2)$, where "o" at the leaf denotes a good path.

$P(r_1, r_2, \ldots, r_i)$. 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

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

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 (eg 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 $|E|/2$, which we now phrase in a way suitable for derandomization.

Algorithm 3.16 (randomized MaxCut, rephrased).
Input: a graph $G = ([n], E)$

Flip n coins $r_1, r_2, \ldots, r_n$, put vertex i in S if $r_i = 1$ and in T if $r_i = 0$. 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 $S_i \stackrel{\text{def}}{=} \{j : j \leq i, r_j = 1\}$ (resp. $T_i \stackrel{\text{def}}{=} \{j : j \leq i, r_j = 0\}$) 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 $G = ([n], E)$

On input $G = ([n], E)$,

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

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 our first motivating example, 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 $S \subseteq [k]$, 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 $S \neq T \subseteq [k]$. 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). 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 $\text{poly}(n)$ time. It is slower than the greedy algorithm obtained by the Method of Conditional Expectations, but it has the advantage of using only $O(\log n)$ workspace and being parallelizable.

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 $\{0,1\}^m = [M]$. The naïve approach is to repeat the above algorithm for the individual bits m times. This uses $(\log M)(\log N)$ bits to start with, 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 sequences 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 $H \xleftarrow{R} \mathcal{H}$ 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. Recall that there is a (unique) finite field $\mathrm{GF}(p^t)$ of size p^t for every prime p and $t \in \mathbb{N}$. It is known how to deterministically construct a description of such a field (i.e. an irreducible polynomial of degree t over $\mathrm{GF}(p) = \mathbb{Z}_p$) in time $\mathrm{poly}(p, t)$. This satisfies our definition of explicitness when the prime p (the *characteristic* of the field) is small, in particular when $p = 2$. Thus, we have 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 $\{0,1\}^m$ 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 (which turns out to be optimal) by using $(ax)|_m + b$ where $b \in \{0,1\}^m$ instead of $(ax + b)|_m$. Summarizing:

Theorem 3.25. 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.

3.5.3 Hash Tables

The original motivating application for pairwise independent functions was for hash tables. Suppose we want to store a set $S \subseteq [N]$ 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 from $h : [N] \to [M]$ for some M to be determined later. Let the table T be of size M. 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, the probability that there are any collisions is

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

for $M = |S|^2/\varepsilon$. 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 3.25, we can generate and store H using $O(\log N)$ random bits. The storage required for the table T is $O(M \log N) = O(|S|^2 \log N)$. The space complexity can be improved to $O(|S| \log N)$, which is nearly optimal for small S, by taking $M = O(|S|)$ 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 need $O(km)$ random bits after error reduction. Here we will see how to reduce the number of random bits required for error reduction by doing only pairwise independent repetitions.

To analyze this, we will need an analogue of the Chernoff Bound 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 3.26 (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 sums of pairwise independent random variables are concentrated around their expectation.

Proposition 3.27 (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 3.27 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 have 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 a constant error that uses m random bits, we have another **BPP** algorithm that uses $O(m + \log n) = O(m)$ random bits

and has an error of $1/\text{poly}(n)$. 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 4.7.)

Sampling. Recall the SAMPLING problem: Given an oracle 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. 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	$O(1/(\varepsilon^2\delta))$	$O(m + \log(1/\varepsilon) + \log(1/\delta))$

3.5.5 k-wise Independence

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

Definition 3.28 (k-wise independent hash functions). For $k \in \mathbb{N}$, 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 3.29 (k-wise independence from polynomials). Let $\mathbb{F}$ be a finite field. Define the family of functions $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_k} : \mathbb{F} \to \mathbb{F}\}$ where each $h_{a_0,a_1,\ldots,a_{k-1}}(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$ for $a, b \in \mathbb{F}$.

Proposition 3.30. The family $\mathcal{H}$ given in Construction 3.29 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 3.31. 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 $\text{poly}(n, m, k)$.

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 3.27, with the key improvement being that the error probability vanishes linearly in $t^{k/2}$ rather than t.

3.6 Exercises

~~Problem 3.1. (Derandomizing RP versus BPP) Show that prRP = prP implies that prBPP = prP~~, and thus also that $\mathbf{BPP} = \mathbf{P}$. (Hint: Look at the proof that $\mathbf{NP} = \mathbf{P} \Rightarrow \mathbf{BPP} = \mathbf{P}$.)

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 $S_1, S_2, \ldots, S_m \subseteq [d]$ is called an (ℓ, a)-*design* 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 $m < \binom{d}{a} / \binom{\ell}{a}^2$, then there exists an (ℓ, a)-design $S_1, \ldots, S_m \subseteq [d]$.
Hint: Use the Probabilistic Method. Specifically, show that if the sets are chosen randomly, then for every $S_1, \ldots, S_{i-1}$,

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

(2) Conclude that for every $\epsilon > 0$, there is a constant c_ϵ such that for all ℓ, there is a design with $a \leq \epsilon\ell$, $m \geq 2^{\epsilon\ell}$, and $d \leq c_\epsilon \ell$. That is, in a universe of size $O(\ell)$, we can fit *exponentially many* sets of size ℓ whose intersections are an arbitrarily small constant fraction of ℓ.
(3) Using the Method of Conditional Expectations, show how to construct designs as in Parts 1 and 2 *deterministically* in time $\text{poly}(m, d)$.

Problem 3.3. (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 $\mathrm{Var}[X_j^2] \leq 2f_2^2$.
(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 3.4. (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?

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. [Sip2, Gol6, AB]. Adleman [Adl] showed that every language in **RP** has polynomial-sized circuits (cf., Corollary 3.12), and 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 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 [Sip2, 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 Luby [Lub2]. For more on this method, see the textbooks [MR, AS].

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 pairwise independence in computer science originates with the seminal papers of Carter and Wegman [CW, WC], which introduced the notions of universal and strongly universal families of hash functions. The pairwise independent and k-wise independent sample spaces of Constructions 3.18, 3.23, and 3.29 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 constructions of pairwise independent hash functions from Problem 3.4 are due to Carter and Wegman [CW]. 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 $O(|S| \log N)$ 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]. Tail bounds for k-wise independent random variables 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 3.3 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* (*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 follows:

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:

- Edge expansion (cuts): instead of $N(S)$, use the number of edges leaving S.
- Random walks: random walks converge quickly to 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).

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 $D - 2$. Fixing a subset S of size K, $N(S)$ is a set of KD random vertices in R (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 multiple edges in the resulting graph can be eliminated or redistributed to obtain a simple graph that is at least as good an expander.
- 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 non-bipartite 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. That is, constructions that are deterministic and efficient. We view 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 is also $(1-\gamma)$ spectral expander. 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 (eg the disjoint union of two good expanders), thereby having no spectral expansion. Also note that the bound on γ depends on D. 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, e.g. $A = D - 1.01$, and clearly one cannot have A to be any larger than D. 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 $1-\lambda$ spectral expander 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:

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 $\lambda(G) \leq 2\sqrt{D-1}/D + o(1)$ 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. 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 is vertex expansion. 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 **??** 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 **??**.

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.

As usual, 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 should be $\langle\chi_S,u\rangle/\|u\| = \sum_i(\chi_S)_i = |S| = \alpha N$. 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

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

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:

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

Thus,

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

To complete the proof, we note that

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

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

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 $\lambda(G)=O\left(\theta\log(1/\theta)\right)$.

Putting the two theorems together, we see that λ and θ differ by at most 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 definition, 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 last time. 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, last time when proving the Expander Mixing Lemma, we expressed the fraction of edges between sets S and T as $\chi_S^t M \chi_T$ (up to some normalization factor). Then we can proceed in one of the two following ways:

Vector Decomposition Decompose the input vector 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 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 $c \in \mathbb{R}$. From 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 $\|E\| \leq 1$.

This lemma has a nice intuition: 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. 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. An example of using P would be to say that 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 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 random walk is entirely within B for the first $\ell+1$ steps and is at node i at step $\ell+1$. 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 random walk is entirely within B for the first $\ell+1$ steps and is at node i at step $\ell+2$. 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 random walk is at i after $\ell+2$ steps, and never leaves B. □

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 $y = xP$. Note that $\|y\| \leq \|x\|$ and y has at most μN

nonzero coordinates. Then

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

so

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

Thus,

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

□

Using Claim 4.21, the probability of never leaving B in a t-step random walk is

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

□

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 on N vertices with 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 $\varepsilon > 0$

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

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 to 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	$O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$	$O(\frac{m}{\varepsilon^2}\log\frac{1}{\delta})$
Pairwise Independent Samples	$O(\frac{1}{\varepsilon^2\delta})$	$O(m + \log\frac{1}{\varepsilon} + \log\frac{1}{\delta})$
Expander Walks	$O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$	$m + O((\log\frac{1}{\delta}) \cdot (\log\frac{1}{\varepsilon})/\varepsilon^2)$

The $\log(1/\varepsilon)$ factor in the number of random bits used by expander walks 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.

Before we end, we make an important remark: we have not actually given an algorithm for randomness-efficient error reduction! 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 implicitly construct an expander (without writing it down), and 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 (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 most combinatorial measures of expansion (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 fundamental 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 4.23 (Gabber–Galil expanders). This is the graph $G = (V, E)$ with nodes $V = [M]^2$ (namely, each node is a pair of numbers from $\{0, ..., M-1\}$), and edges connecting each node (x, y) with each of the following nodes: $(x, y), (x, x+y), (x, x+y+1), (x+y, y)$, and $(x+y+1, y)$, where all arithmetic is mod M.

This is a fully explicit 5-regular digraph with $N = M^2$ nodes and spectral expansion $\gamma = \Omega(1)$, proved using Fourier analysis. We note that this graph is directed, but it can be made undirected by adding a reverse copy of each edge.

Construction 4.24 (p-Cycle with inverse chords). This is the graph $G = (V, E)$ with vertex set $V = \mathbb{Z}_p$, the integers modulo 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 4.25 (Ramanujan graphs). The graph $G = (V, E)$ with nodes $V = \mathbb{F}_q \cup \{\infty\}$ elements in 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 $p \neq q$ satisfying $p \equiv 1 \bmod 4$ and where $i^2 = -1 \bmod q$.

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.

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 $\gamma \geq 2\sqrt{D-1}/D$.

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

4.3.2.1 Squaring

~~**Definition 4.27 (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 4.28. 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 4.29 (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 $D_1 D_2$-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 4.30 (Tensor Products of Vectors and Matrices). Let $x \in \mathbb{R}^{N_1}, y \in \mathbb{R}^{N_2}$, then their *tensor product* $z = x \otimes y \in \mathbb{R}^{N_1 N_2}$ is the vector[2] $z_{ij} = x_i y_j$.

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

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 $[N_1] \times [N_2]$ elements 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.

[2] For convenience we index the vector z by two indices i, j. To transform to a standard indexing we can map ij to $iN_1 + j$.

- Not all vectors $z \in \mathbb{R}^{N_1 N_2}$ 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 $\mathbb{R}^{N_1 N_2}$.
- 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.

Lemma 4.31. 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 $(N_1 N_2, D_1 D_2, \min\{\gamma_1, \gamma_2\})$-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 for the construction 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[3] 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 on the clouds must be far from uniform, *or*
(2) the conditional distribution 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 instead 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 $x \in \mathbb{R}^{N_1 N_2}$ that is orthogonal to $u_{N_1 N_2}$, we can decompose x as $x = x^{\parallel} + x^{\perp}$ where $x^{\parallel}$ 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 $x^{\parallel} = y \otimes u_{N_2}$, where $y \in \mathbb{R}^{N_1}$ is orthogonal to u_{N_1} (because $x^{\parallel} = x - x^{\perp}$ is orthogonal to $u_{N_1 N_2}$). If we think of x as the nonuniform component of a probability distribution, then $x^{\parallel}$ and $x^{\perp}$ correspond to the two cases in the intuition above.

For the first case, we have

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

[3] 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_1 tells us that M_1 shrinks y by a factor of λ_1, and thus M shrinks $x^{\|}$ by the same factor. For the second case, we write

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

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 $x^{\|}M$ and $x^{\perp}M$ are orthogonal. Note that $x^{\|}M = (yM_1) \otimes u_{N_2}$ 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,

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

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 random step on G, but using a random steps 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 an 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 symmetrical. 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 4.32 (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 $G ⓩ H$ 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 v. Let j' be such that e is the j''th edge entering v in G. (In an undirected graph, this means that u is the j''th neighbor of v.)
(3) Let j be the b'th neighbor of j' in H.

Theorem 4.33. If G is a (N_1, D_1, γ_1)-graph, and H is a (D_1, D_2, γ_2)-graph then $G ⓩ H$ 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 on the vertices of $G_1 ⓩ G_2$ that is far from uniform, there are two extreme cases just as in the intuition for the tensor product. Either
  (1) All the (conditional) distributions within the clouds are far from uniform, *or*
  (2) All the (conditional) distributions within the clouds of size D_1 are uniform (in which case the marginal distribution on the clouds must be far from uniform).

  In Case 1, the first H-step 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 has the effect of making the marginal distribution on clouds closer to uniform. But note that we haven't actually gotten closer to the uniform distribution on the vertices of $G_1 ⓩ G_2$ because the G-step is a permutation. Still, if the marginal distribution on clouds has become closer to uniform, then the conditional distributions within the clouds must have become further from uniform, and thus the second H-step 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 (unlike the case of the tensor product, where we were able to show that $x^{\|}M$ is orthogonal to $x^{\perp}M$).
- 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 4.33] 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

~~Construction 4.34 (Mildly Explicit Expanders).~~ ~~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: Let H be a $(D^4, D, 1 - \lambda_0)$-graph for $\lambda_0 = 1/8$ (e.g., as constructed in Problem 4.3).

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

Proposition 4.35. 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 4.28, $G_1 = H^2$ 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,

$$\begin{aligned} \deg(G_{t+1}) &= \deg(H)^2 = D^2 \\ \#\text{nodes}(G_{t+1}) &= \#\text{nodes}(G_t^2) \cdot \#\text{nodes}(H) = N_t \cdot D^4 = D^{4t} D^4 = D^{4(t+1)} \\ \lambda(G_{t+1}) &\leq \lambda(G_t)^2 + 2\lambda_0 \leq (1/2)^2 + 2 \cdot (1/8) = 1/2 \end{aligned}$$

□

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 actually 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.

Construction 4.36 (Fully Explicit Expanders). We remedy the above difficulty by using tensoring to make the sizes of the graphs grow more quickly. Let H be a $(D^8, D, 1/8)$-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 grow doubly exponentially $N_t \approx c^{2^t}$, while 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 4.37. There is a constant D such that for every $t \in \mathbb{N}$, there is a fully explicit expander graph G_t with degree D, spectral expansion $7/8$, and $N_t = D^{4t}$ nodes.

4.3.4 Open Problems

As we have seen, spectral expanders such as those in Theorem 4.37 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 explicit constructions of optimal spectral expanders, namely Ramanujan graphs. However, unlike the expanders of Theorem 4.37, 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 4.38. 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 5), 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 4.39. 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 4.40. 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 the case of highly imbalanced bipartite expanders is even harder (despite them being useful in a number of applications); see 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 4.41 (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 2.55. 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 4.28, 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 4.33, we have

$$\begin{aligned} \gamma(C_{k-1}^2 \textcircled{z} H) &\geq \gamma(H)^2 \cdot \gamma(C_{k-1}^2) \\ &\geq \left(\frac{3}{4}\right)^2 \cdot 2 \cdot \gamma(C_{k-1}) \cdot (1 - \gamma(C_{k-1})/2)) \\ &\geq \min\left\{\frac{17}{16} \cdot \gamma(C_{k-1}), \frac{1}{18}\right\} \end{aligned}$$

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

$$\gamma(C_k) \geq \min\left\{\frac{17}{16} \cdot \gamma(C_{k-1}), \frac{1}{18}\right\}.$$

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($G_{k-1}^2 \textcircled{z} H$) $=$ space(G_{k-1}^2)$+ O(1)$. For G_{k-1}^2, we are given a triple $(v, (i_1, i_2))$, 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 the tape contents are now (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), which is the final output. For $G_{k-1}^2 \textcircled{z} H$, 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^2_{k-1}), 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^2_{k-1}.

Thus we have proven:

Theorem 4.42. 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 4.42 is still quite recent (2005), there is a good chance that we have not yet exhausted the ideas in it.

Open Problem 4.43. Show that $\mathbf{RL} \subseteq \mathbf{L}^c$ 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 4.44 (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 4.42 to prove **RL** = **L**.

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 λN, 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 $1/\lambda$, 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. (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 4.4. (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 γ_1 and G_2 has spectral expansion γ_2, 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) Prove that a dependence on D_2 in Part 1 is necessary by showing that $\gamma(G_1 \textcircled{r} G_2) = O(1/D_2)$ for sufficiently large N_1.

Problem 4.5. (Bounds on the Zig-Zag Product) In this problem, you will show some limitations on the zig-zag product, namely that its expansion is limited by that of the two composed graphs. Show that:

(1) The vertex expansion of $G ⓩ H$ is at most the degree of H. (That is, find a "small" set S which does not expand by more than the degree of H. By small, we mean that $|S|$ should be sublinear in the number of vertices of $G ⓩ H$.)
(2) $\lambda(G ⓩ H) \geq \lambda(G)$.
(3) $\lambda(G ⓩ H) \geq \lambda(H)^2$ if the labelling of clouds is consistent, i.e. if $e = (u, v)$ is the i'th edge leaving u, then it is also the i'th edge leaving v. (In other words, the same vertex of H is mapped to the two vertices of $G ⓩ H$ corresponding to e (one in cloud u and one in cloud v).)

Problem 4.6. (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 1-bit probes.

However, note that such bipartite vertex expanders do *not* follow from explicit spectral expanders as given in Theorem 4.37, 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 later in this survey, we will see how to explicitly construct expanders that are quite good for this application (specifically, with $M = K^{1.01} \cdot \text{polylog}N$).

Problem 4.7. (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.

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. A 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.

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 expansion roughly $D/2$ for small sets.

Forms of the Expander Mixing Lemma date back to Alon and Chung [AC2], who considered the case that $T = S^c$. The converse to the Expander Mixing Lemma (Theorem 4.16) is due to Bilu and Linac [BL]. For more on quasirandomness, see [CGW, AS] 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 [Gil2]. Our proof of the Chernoff Bound is inspired by that of Healy [Hea], who also provides some other variants and generalizations. Problem 4.7 is due to Karp, Pippenger, and Sipser [KPS], who initiated the study of randomness-efficient error reduction of randomized algorithms.

The first explicit construction of constant-degree expanders was given by Margulis [Mar1], albeit with a nonconstructive proof that the spectral expansion is bounded away from 0. Construction 4.23 is due to Gabber and Galil [GG], who simplified Margulis' construction and gave an elementary proof of expansion, with an explicit bound on the second eigenvalue. Ramanujan graphs (Construction 4.25) 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 [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 4.4, on the replacement product, is from [RVW], and can be used in place of the zig-zag product in both the expander constructions and the UNDIRECTED S-T CONNECTIVITY algorithm (Algorithm 4.41. 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 Open Problem 4.38. 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 Capalbo et al. [CRVW]. Alon and Capalbo [AC1] have made progress on Open Problem 4.40 by giving an explicit construction of nonbipartite constant-degree "unique-neighbor" expanders (see Problem 4.6).

The deterministic logspace algorithm for UNDIRECTED S-T CONNECTIVITY (Algorithm 4.41) is due to Reingold [Rei]. Saks and Zhou [SZ] showed that $\mathbf{RL} \subseteq \mathbf{L}^{3/2}$, making nontrivial use of Nisan's pseudorandom generator for space-bounded computation [Nis]. This was slightly improved by Armoni [Arm], who showed that $\mathbf{RL} \subseteq \mathbf{DSPACE}((\log^{3/2} n)/\sqrt{\log\log n})$, which remains the best known derandomization of **RL**.

Based on Algorithm 4.41, 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 4.3, Part 1 is a variant of a construction of Alon [Alo2]; Part 2 is from [RVW]. The result of Problem 4.6, on bit-probe data structures for set membership, is due to Buhrman, Miltersen, Radhakrishnan, and Venkatesan [BMRV].

Preview of Volume II

Volume II will contain a thorough treatment of three other major "pseudorandom objects" of this survey — randomness extractors, list-decodable error-correcting codes, and pseudorandom generators. In addition to developing the basic theory and providing constructions for each, a major focus will be the connections between each of these objects have to each other and to expander graphs. The survey will conclude by showing how all four pseudorandom objects can be cast in a single "list-decoding" framework, with clarifies both the similarities and differences between them. It will also include a brief discussion of some significant topics omitted from this survey, with pointers to the literature.

Tentatively, the outline of Volume II will be as follows (omitting exercises and references, which will be included in each chapter):

- Randomness Extractors
  - Motivation and Definitions
  - Relation to Hashing and Expanders
  - Constructing Extractors
  - Lossless Condensers as Expanders
- List-Decodable Codes
  - Motivation and Definitions
  - List-Decoding Algorithms
  - List-Decoding View of Expanders and Extractors
- Pseudorandom Generators
  - Motivation and Definitions
  - Survey of Cryptographic Pseudorandom Generators
  - Pseudorandom Generators from Average-Case Hardness

  - Worst-Case vs. Average-Case Hardness and Locally (List-)Decodable Codes
  - Black-Box PRG Constructions and Extractors
- Conclusions
  - A Unified Theory
  - Other Topics

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 *Proceedings of the Nineteenth 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 *FOCS*, pages 73–. IEEE Computer Society, 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).

[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. Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996).

[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.

[AS] 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.

[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. *Complexity Theory: A Modern Approach*. Cambridge University Press, 2008. To appear.

[Art] M. Artin. *Algebra*. Prentice Hall Inc., Englewood Cliffs, NJ, 1991.

[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. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan Sparsifiers. *CoRR*, abs/0808.0163, 2008.

[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 R. E. Ladner and C. Dwork, editors, *STOC*, pages 325–334. ACM, 2008.

[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.

[Bro] A. Z. Broder. How hard is to marry at random? (On the approximation of the permanent). In *STOC*, 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.

[CRVW] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness Conductors and Constant-Degree Lossless Expanders. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 659–668, Montréal, CA, May 2002. ACM. In 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.

[CGW] F. R. K. Chung, R. L. Graham, and R. M. Wilson. Quasi-random graphs. *Combinatorica*, 9(4):345–362, 1989.

[CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[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.

[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.

[Eli] P. Elias. *List decoding for noisy channels*. Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Mass., Rep. No. 335, 1957.

[Erd] P. Erdös. 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.

[ES] P. Erdős and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory. Series A*, 14:298–301, 1973.

[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.

[Fil] J. A. Fill. Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the exclusion process. *The Annals of Applied Probability*, 1(1):62–87, 1991.

[FKS] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 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.

[Gil1] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[Gil2] D. Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 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 Association for Computing Machinery*, 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, 2008. To appear.

[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 Association for Computing Machinery*, 38(3):691–729, 1991.

[GM] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.

[Gur] V. Guruswami. *Algorithmic Results in List Decoding*, volume 2, number 2 of *Foundations and Trends in Theoretical Computer Science*. now publishers, 2006.

[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 *FOCS*, pages 531–542. IEEE Computer Society, 2006.

[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.

[HLW] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the AMS*, 43(4):439–561, 2006.

[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.

[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.

[Kah] N. Kahale. Eigenvalues and expansion of regular graphs. *Journal of the Association for Computing Machinery*, 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 and Hall/CRC Press, 2007. To appear.

[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 *Proceedings of the Thirty-Third 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*. Morgan Kaufmann, San Mateo, CA, 1992. Arrays, trees, hypercubes.

[LV] D. Lewin and S. Vadhan. Checking polynomial identities over any field: towards a derandomization? In *STOC '98 (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 *FCT*, pages 565–574, 1979.

[Lov2] 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. *J. Comput. System Sci.*, 47(2):250–286, 1993. 29th Annual IEEE Symposium on Foundations of Computer Science (White Plains, NY, 1988).

[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.

[MS] 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.

[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 Peredachi Informatsii*, 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 *FOCS*, 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 *FOCS*, pages 248–255. IEEE Computer Society, 2004.

[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 Math.*, 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.

[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 *FOCS*, pages 307–311. IEEE, 1979.

[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 *FOCS*, pages 4–. IEEE Computer Society, 2003.

[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 Association for Computing Machinery*, 23(3):555–565, 1976.

[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. 36th IEEE Symposium on the Foundations of Computer Science (Milwaukee, WI, 1995).

[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 Association for Computing Machinery*, 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 *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 330–335, Boston, Massachusetts, 25–27 Apr. 1983.

[Sip2] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2nd edition, 2005.

[SS] 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.

[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. Essential Coding Theory (Lecture Notes). `http://people.csail.mit.edu/madhu/FT04/`, 2004.

[Tan] M. R. Tanner. Explicit Concentrators from Generalized N-gons. *SIAM Journal on Algebraic Discrete Methods*, 5(3):287–293, 1984.

[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. Working papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N. M., 1975).

[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. Special issue dedicated to Michael Machtey.

[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.