Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007

`Comments welcome!`

Sanjeev Arora and Boaz Barak
Princeton University
`complexitybook@gmail.com`

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to `complexitybook@gmail.com` — **Thank You!!**

Chapter 7

Randomized Computation

"We do not assume anything about the distribution of the instances of the problem to be solved. Instead we incorporate randomization into the algorithm itself... It may seem at first surprising that employing randomization leads to efficient algorithm. This claim is substantiated by two examples. The first has to do with finding the nearest pair in a set of n points in $\mathbb{R}^k$. The second example is an extremely efficient algorithm for determining whether a number is prime."
Michael Rabin, 1976

Thus far our standard model of computation has been the deterministic Turing Machine. But everybody who is even a little familiar with computation knows that that real-life computers need not be deterministic since they have built-in "random number generators." In fact these generators are very useful for computer simulation of "random" processes such as nuclear fission or molecular motion in gases or the stock market. This chapter formally studies *probablistic computation*, and complexity classes associated with it.

We should mention right away that it is an open question whether or not the universe has any randomness in it (though quantum mechanics seems to guarantee that it does). Indeed, the output of current "random number generators" is not guaranteed to be truly random, and we will revisit this limitation in Section 7.4.3. For now, assume that true random number generators exist. Then arguably, a realistic model for a real-life computer is a Turing machine with a random number generator, which we call a *Probabilistic Turing Machine* (PTM). It is natural to wonder whether difficult problems like 3SAT are efficiently solvable using a PTM.

We will formally define the class **BPP** of languages decidable by polynomial-time PTMs and discuss its relation to previously studied classes such as **P**/poly and **PH**. One consequence is that if **PH** does not collapse, then 3SAT does not have efficient probabilistic algorithms.

We also show that probabilistic algorithms can be very practical by presenting ways to greatly reduce their error to absolutely minuscule quantities. Thus the class **BPP** (and its sister classes **RP**, **coRP** and **ZPP**) introduced in this chapter are arguably as important as **P** in capturing efficient computation. We will also introduce some related notions such as probabilistic logspace algorithms and probabilistic reductions.

Though at first randomization seems merely a tool to allow simulations of randomized physical processes, the surprising fact is that in the past three decades randomization has led to more efficient —and often simpler—algorithms for problems in a host of other fields—such as combinatorial optimization, algebraic computation, machine learning, and network routing.

In complexity theory too, the role of randomness extends far beyond a study of randomized algorithms and classes such as **BPP**. Entire areas such as cryptography and interactive and probabilistically checkable proofs rely on randomness in an essential way, sometimes to prove results whose statement did not call for randomness at all. The groundwork for studying those areas will be laid in this chapter.

In a later chapter, we will learn something intriguing: to some extent, the power of randomness may be a mirage. If a certain plausible complexity-theoretic conjecture is true (see Chapters 16 and 17), then every probabilistic algorithm can be simulated by a deterministic algorithm (one that does not use any randomness whatsoever) with only polynomial overhead.

Throughout this chapter and the rest of the book, we will use some notions from elementary probability on finite sample spaces; see Appendix A for a quick review.

7.1 Probabilistic Turing machines

We now define probabilistic Turing machines (PTMs). Syntactically, a PTM is no different from a nondeterministic TM: it is a TM with two transition functions δ_0, δ_1. The difference lies in how we interpret the graph of all possible computations: instead of asking whether there *exists* a sequence of choices that makes the TM accept, we ask how large is the *fraction* of choices for which this happens. More precisely, if M is a PTM, then we envision that in every step in the computation, M chooses randomly which one of its transition functions to apply (with probability half applying δ_0 and with probability half applying δ_1). We say that M decides a language if it outputs the right answer with probability at least $2/3$.

Notice, the ability to pick (with equal probability) one of δ_0, δ_1 to apply at each step is equivalent to the machine having a "fair coin", which, each time it is tossed, comes up "Heads" or "Tails" with equal probability *regardless of the past history of Heads/Tails.* As mentioned, whether or not such a coin exists is a deep philosophical (or scientific) question.

Definition 7.1 (The classes **BPTIME** and **BPP**)
For $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that a PTM M decides L in time $T(n)$, if for every $x \in \{0,1\}^*$, M halts in $T(|x|)$ steps regardless of its random choices, and $\Pr[M(x) = L(x)] \geq 2/3$, where we denote $L(x) = 1$ if $x \in L$ and $L(x) = 0$ if $x \notin L$. We let $\mathbf{BPTIME}(T(n))$ denote the class of languages decided by PTMs in $O(T(n))$ time and let $\mathbf{BPP} = \cup_c \mathbf{BPTIME}(n^c)$.

Remark 7.2
We will see in Section 7.4 that this definition is quite robust. For instance, the "coin" need not be fair. The constant $2/3$ is arbitrary in the sense that it can be replaced with any other constant

greater than half without changing the classes $\mathbf{BPTIME}(T(n))$ and **BPP**. Instead of requiring the machine to always halt in polynomial time, we could allow it to halt in *expected* polynomial time.

Remark 7.3
While Definition 7.1 allows the PTM M, given input x, to output a value different from $L(x)$ with positive probability, this probability is only over the random choices that M makes in the computation. In particular for *every* input x, $M(x)$ will output the right value $L(x)$ with probability at least $2/3$. Thus **BPP**, like **P**, is still a class capturing *worst-case* computation.

Since a deterministic TM is a special case of a PTM (where both transition functions are equal), the class **BPP** clearly contains **P**. As alluded above, under plausible complexity assumptions it holds that **BPP** = **P**. Nonetheless, as far as we know it may even be that **BPP** = **EXP**. (Note that $\mathbf{BPP} \subseteq \mathbf{EXP}$, since given a polynomial-time PTM M and input $x \in \{0,1\}^n$ in time $2^{\mathrm{poly}(n)}$ it is possible to enumerate all possible random choices and compute precisely the probability that $M(x) = 1$.)

An alternative definition. As we did with **NP**, we can define **BPP** using deterministic TMs where the "probabilistic choices" to apply at each step can be provided to the TM as an additional input:

Definition 7.4 (**BPP**, alternative definition)
BPP contains a language L if there exists a polynomial-time TM M and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$, $\Pr_{r \in_R \{0,1\}^{p(|x|)}}[M(x,r) = L(x)] \geq 2/3$.

7.2 Some examples of PTMs

The following examples demonstrate how randomness can be a useful tool in computation. We will see many more examples in the rest of this book.

7.2.1 Probabilistic Primality Testing

In *primality testing* we are given an integer N and wish to determine whether or not it is prime. Generations of mathematicians have learnt about prime numbers and —before the advent of computers— needed to do primality testing to test various conjectures[1]. Ideally, we want efficient algorithms, which run in time polynomial in the size of N's representation, in other words, $\mathrm{poly}(\log n)$. We knew of no such efficient algorithms[2] until the 1970s, when an effficient probabilistic algorithm was discovered. This was one of the first to demonstrate the power of probabilistic algorithms. In a recent breakthrough, Agrawal, Kayal and Saxena [**?**] gave a *deterministic* polynomial-time algorithm for primality testing.

[1]Though a very fast human computer himself, Gauss used the help of a human supercomputer –an autistic person who excelled at fast calculations—to do primality testing.

[2]In fact, in his letter to von Neumann quoted in Chapter 2, Gödel explicitly mentioned this problem as an example for an interesting problem in **NP** but not known to be efficiently solvable.

Formally, primality testing consists of checking membership in the language $\mathsf{PRIMES} = \{ \llcorner N \lrcorner : N$ is a prime num
Notice, the corresponding *search* problem of finding the factorization of a given composite number N seems very different and much more difficult. It is the famous FACTORING problem, whose conjectured hardness underlies many current cryptosystems. Chapter 20 describes Shor's algorithm to factors integers in polynomial time in the model of *quantum computers*.

We sketch an algorithm showing that PRIMES is in **BPP** (and in fact in **coRP**). For every number N, and $A \in [N-1]$, define

$$QR_N(A) = \begin{cases} 0 & gcd(A, N) \neq 1 \\ +1 & \begin{array}{l} A \text{ is a } \textit{quadratic residue} \text{ modulo } N \\ \text{That is, } A = B^2 \pmod{N} \text{ for some } B \text{ with } gcd(B,N) = 1 \end{array} \\ -1 & \text{otherwise} \end{cases}$$

We use the following facts that can be proven using elementary number theory:

- For every odd prime N and $A \in [N-1]$, $QR_N(A) = A^{(N-1)/2} \pmod{N}$.
- For every odd N, A define the *Jacobi symbol* $(\frac{N}{A})$ as $\prod_{i=1}^{k} QR_{P_i}(A)$ where $P_1, \ldots, P_k$ are all the (not necessarily distinct) prime factors of N (i.e., $N = \prod_{i=1}^{k} P_i$). Then, the Jacobi symbol is computable in time $O(\log A \cdot \log N)$.
- For every odd composite N, $\left|\{A \in [N-1] : gcd(N,A) = 1 \text{ and } (\frac{N}{A}) = A^{(N-1)/2}\}\right| \leq \frac{1}{2}\left|\{A \in [N-1] : gcd(N,A) = 1\}\right|$

Together these facts imply a simple algorithm for testing primality of N (which we can assume without loss of generality is odd): choose a random $1 \leq A < N$, if $gcd(N,A) > 1$ or $(\frac{N}{A}) \neq A^{(N-1)/2} \pmod{N}$ then output "composite", otherwise output "prime". This algorithm will always output "prime" is N is prime, but if N is composite will output "composite" with probability at least 1/2. (Of course this probability can be amplified by repeating the test a constant number of times.)

7.2.2 Polynomial identity testing

So far we described probabilistic algorithms solving problems that have known deterministic polynomial time algorithms. We now describe a problem for which no such deterministic algorithm is known:

We are given a polynomial with integer coefficients in an implicit form, and we want to decide whether this polynomial is in fact identically zero. We will assume we get the polynomial in the form of an *arithmetic circuit*. This is analogous to the notion of a Boolean circuit, but instead of the operators $\wedge, \vee$ and $\neg$, we have the operators $+, -$ and $\times$. Formally, an n-variable arithmetic circuit is a directed acyclic graph with the sources labeled by a variable name from the set $x_1, \ldots, x_n$, and each non-source node has in-degree two and is labeled by an operator from the set $\{+, -, \times\}$. There is a single sink in the graph which we call the *output* node. The arithmetic circuit defines a polynomial from $\mathbb{Z}^n$ to $\mathbb{Z}$ by placing the inputs on the sources and computing the value of each node using the appropriate operator. We define ZEROP to be the set of arithmetic circuits that compute the identically zero polynomial. Determining membership in ZEROP is also called *polynomial identity testing*, since we can reduce the problem of deciding whether two circuits C, C' compute the same polynomial to ZEROP by constructing the circuit D such that $D(x_1, \ldots, x_n) = C(x_1, \ldots, x_n) - C'(x_1, \ldots, x_n)$.

Since expanding all the terms of a given arithmetic circuit can result in a polynomial with exponentially many monomials, it seems hard to decide membership in ZEROP. Surprisingly, there is in fact a simple and efficient probabilistic algorithm for testing membership in ZEROP. At the heart of this algorithm is the following fact, typically known as the Schwartz-Zippel Lemma, whose proof appears in Appendix A (see Lemma A.25):

Lemma 7.5
Let $p(x_1, x_2, \ldots, x_m)$ be a polynomial of total degree at most d and S is any finite set of integers. When $a_1, a_2, \ldots, a_m$ are randomly chosen with replacement from S, then

$$\Pr[p(a_1, a_2, \ldots, a_m) \neq 0] \geq 1 - \frac{d}{|S|}.$$

Now it is not hard to see that given a size m circuit C on n variables, it defines a polynomial of degree at most 2^m. This suggests the following simple probabilistic algorithm: choose n numbers $x_1, \ldots, x_n$ from 1 to $10 \cdot 2^m$ (this requires $O(n \cdot m)$ random bits), evaluate the circuit C on $x_1, \ldots, x_n$ to obtain an output y and then accept if $y = 0$, and reject otherwise. Clearly if $C \in \mathsf{ZEROP}$ then we always accept. By the lemma, if $C \notin \mathsf{ZEROP}$ then we will reject with probability at least $9/10$.

However, there is a problem with this algorithm. Since the degree of the polynomial represented by the circuit can be as high as 2^m, the output y and other intermediate values arising in the computation may be as large as $(10 \cdot 2^m)^{2^m}$ — this is a value that requires exponentially many bits just to describe!

We solve this problem using the technique of *fingerprinting*. The idea is to perform the evaluation of C on $x_1, \ldots, x_n$ modulo a number k that is chosen at random in $[2^{2m}]$. Thus, instead of computing $y = C(x_1, \ldots, x_n)$, we compute the value $y \pmod k$. Clearly, if $y = 0$ then $y \pmod k$ is also equal to 0. On the other hand, we claim that if $y \neq 0$, then with probability at least $\delta = \frac{1}{10m}$, k does not divide y. (This will suffice because we can repeat this procedure $O(1/\delta)$ times to ensure that if $y \neq 0$ then we find this out with probability at lest $9/10$.) Indeed, assume that $y \neq 0$ and let $\mathcal{S} = \{p_1, \ldots, p_\ell\}$ denote set of the distinct prime factors of y. It is sufficient to show that with probability at δ, the number k will be a prime number not in $\mathcal{S}$. Yet, by the prime number theorem, the probability that k is prime is at least $\frac{1}{5m} = 2\delta$. Also, since y can have at most $\log y \leq 5m2^m$ distinct factors, the probability that k is in $\mathcal{S}$ is less than $\frac{5m2^m}{2^{2m}} \ll \frac{1}{10m} = \delta$. Hence by the union bound, with probability at least δ, k will not divide y.

7.2.3 Testing for perfect matching in a bipartite graph.

If $G = (V_1, V_2, E)$ is the bipartite graph where $|V_1| = |V_2|$ and $E \subseteq V_1 \times V_2$ then a *perfect matching* is some $E' \subseteq E$ such that every node appears exactly once among the edges of E'. Alternatively, we may think of it as a permutation σ on the set $\{1, 2, \ldots, n\}$ (where $n = |V_1|$) such that for each $i \in \{1, 2, \ldots, n\}$, the pair $(i, \sigma(i))$ is an edge. Several deterministic algorithms are known for detecting if a perfect matching exists. Here we describe a very simple randomized algorithm (due to Lovász) using the Schwartz-Zippel lemma.

Consider the $n \times n$ matrix X (where $n = |V_1| = |V_2|$) whose (i,j) entry X_{ij} is the variable x_{ij} if $(i,j) \in E$ and 0 otherwise. Recall that the determinant of matrix $det(X)$ is

$$det(X) = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_{i=1}^{n} X_{i,\sigma(i)}, \tag{1}$$

where S_n is the set of all permutations of $\{1, 2, \ldots, n\}$. Note that every permutation is a potential perfect matching, and the corresponding monomial in $det(X)$ is nonzero iff this perfect matching exists in G. Thus the graph has a perfect matching iff $det(X) \neq 0$.

Now observe two things. First, the polynomial in (1) has $|E|$ variables and total degree at most n. Second, even though this polynomial may be of exponential size, for every setting of values to the X_{ij} variables it can be efficiently evaluated, since computing the determinant of a matrix with integer entries is a simple polynomial-time computation (actually, even in $\mathbf{NC}^2$).

This leads us to Lovász's randomized algorithm: pick random values for X_{ij}'s from $[1, \ldots, 2n]$, substitute them in X and compute the determinant. If the determinant is nonzero, output "accept" else output "reject." The advantage of the above algorithm over classical algorithms is that it can be implemented by a randomized **NC** circuit, which means (by the ideas of Section 6.5.1) that it has a fast implementation on parallel computers.

7.3 One-sided and zero-sided error: RP, coRP, ZPP

The class **BPP** captured what we call probabilistic algorithms with *two sided* error. That is, it allows the machine M to output (with some small probability) both 0 when $x \in L$ and 1 when $x \notin L$. However, many probabilistic algorithms have the property of *one sided* error. For example if $x \notin L$ they will *never* output 1, although they may output 0 when $x \in L$. This is captured by the definition of **RP**.

DEFINITION 7.6
$\mathbf{RTIME}(t(n))$ contains every language L for which there is a is a probabilistic TM M running in $t(n)$ time such that

$$x \in L \Rightarrow \mathbf{Pr}[M \text{ accepts } x] \geq \frac{2}{3}$$
$$x \notin L \Rightarrow \mathbf{Pr}[M \text{ accepts } x] = 0$$

We define $\mathbf{RP} = \cup_{c>0} \mathbf{RTIME}(n^c)$.

Note that $\mathbf{RP} \subseteq \mathbf{NP}$, since every accepting branch is a "certificate" that the input is in the language. In contrast, we do not know if $\mathbf{BPP} \subseteq \mathbf{NP}$. The class $\mathbf{coRP} = \{L \mid \overline{L} \in \mathbf{RP}\}$ captures one-sided error algorithms with the error in the "other direction" (i.e., may output 1 when $x \notin L$ but will never output 0 if $x \in L$).

For a PTM M, and input x, we define the random variable $T_{M,x}$ to be the running time of M on input x. That is, $\Pr[T_{M,x} = T] = p$ if with probability p over the random choices of M on input x, it will halt within T steps. We say that M has *expected running time* $T(n)$ if the expectation $\mathsf{E}[T_{M,x}]$ is at most $T(|x|)$ for every $x \in \{0,1\}^*$. We now define PTMs that never err (also called "zero error" machines):

Definition 7.7
The class **ZTIME**$(T(n))$ contains all the languages L for which there is an expected-time $O(T(n))$ machine that never errs. That is,

$$x \in L \Rightarrow \mathbf{Pr}[M \text{ accepts } x] = 1$$
$$x \notin L \Rightarrow \mathbf{Pr}[M \text{ halts without accepting on } x] = 1$$

We define $\mathbf{ZPP} = \cup_{c>0}\mathbf{ZTIME}(n^c)$.

The next theorem ought to be slightly surprising, since the corresponding statement for nondeterminism is open; i.e., whether or not $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$.

Theorem 7.8
$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.

We leave the proof of this theorem to the reader (see Exercise 4). To summarize, we have the following relations between the probabilistic complexity classes:

$$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$$
$$\mathbf{RP} \subseteq \mathbf{BPP}$$
$$\mathbf{coRP} \subseteq \mathbf{BPP}$$

7.4 The robustness of our definitions

When we defined **P** and **NP**, we argued that our definitions are robust and were likely to be the same for an alien studying the same concepts in a faraway galaxy. Now we address similar issues for probabilistic computation.

7.4.1 Role of precise constants, error reduction.

The choice of the constant $2/3$ seemed pretty arbitrary. We now show that we can replace $2/3$ with any constant larger than $1/2$ and in fact even with $1/2 + n^{-c}$ for a constant $c > 0$.

Lemma 7.9
For $c > 0$, let $\mathbf{BPP}_{n^{-c}}$ denote the class of languages L for which there is a polynomial-time PTM M satisfying $\Pr[M(x) = L(x)] \geq 1/2 + |x|^{-c}$ for every $x \in \{0,1\}^*$. Then $\mathbf{BPP}_{n^{-c}} = \mathbf{BPP}$.

Since clearly $\mathbf{BPP} \subseteq \mathbf{BPP}_{n^{-c}}$, to prove this lemma we need to show that we can transform a machine with success probability $1/2 + n^{-c}$ into a machine with success probability $2/3$. We do this by proving a much stronger result: we can transform such a machine into a machine with success probability exponentially close to one!

Theorem 7.10 (Error reduction)
Let $L \subseteq \{0,1\}^*$ be a language and suppose that there exists a polynomial-time PTM M such that for every $x \in \{0,1\}^*$, $\Pr[M(x) = L(x) \geq \frac{1}{2} + |x|^{-c}$.
Then for every constant $d > 0$ there exists a polynomial-time PTM M' such that for every $x \in \{0,1\}^*$, $\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$.

Proof: The machine M' is quite simple: *for every input $x \in \{0,1\}^*$, run $M(x)$ for k times obtaining k outputs $y_1, \ldots, y_k \in \{0,1\}$, where $k = 8|x|^{2d+c}$. If the majority of these values are* 1 *then accept, otherwise reject.*

To analyze this machine, define for every $i \in [k]$ the random variable X_i to equal 1 if $y_i = L(x)$ and to equal 0 otherwise. Note that $X_1, \ldots, X_k$ are independent Boolean random variables with $\mathsf{E}[X_i] = \Pr[X_i = 1] \geq 1/2 + n^{-c}$ (where $n = |x|$). The Chernoff bound (see Theorem A.18 in Appendix A) implies the following corollary:

Corollary 7.11
Let $X_1, \ldots, X_k$ be independent identically distributed Boolean random variables, with $\Pr[X_i = 1] = p$ for every $1 \leq i \leq k$. Let $\delta \in (0,1)$. Then,

$$\Pr\Big[\Big|\tfrac{1}{k}\sum_{i=1}^{k} X_i - p\Big| > \delta\Big] < e^{-\frac{\delta^2}{4}pk}$$

In our case $p = 1/2 + n^{-c}$, and plugging in $\delta = n^{-c}/2$, the probability we output a wrong answer is bounded by

$$\Pr[\tfrac{1}{n}\sum_{i=1}^{k} X_i \leq 1/2 + n^{-c}/2] \leq e^{-\frac{1}{4n^{-2c}}\frac{1}{2}8n^{2c+d}} \leq 2^{-n^d}$$

■

A similar result holds for the class **RP**. In fact, there we can replace the constant 2/3 with every positive constant, and even with values as low as n^{-c}. That is, we have the following result:

Theorem 7.12
*Let $L \subseteq \{0,1\}^*$ such that there exists a polynomial-time PTM M satisfying for every $x \in \{0,1\}^*$: **(1)** If $x \in L$ then $\Pr[M(x) = 1)] \geq n^{-c}$ and **(2)** if $x \notin L$, then $\Pr[M(x) = 1] = 0$.*

*Then for every $d > 0$ there exists a polynomial-time PTM M' such that for every $x \in \{0,1\}^*$, **(1)** if $x \in L$ then $\Pr[M'(x) = 1] \geq 1 - 2^{-n^d}$ and **(2)** if $x \notin L$ then $\Pr[M'(x) = 1] = 0$.*

These results imply that we can take a probabilistic algorithm that succeeds with quite modest probability and transform it into an algorithm that succeeds with overwhelming probability. In fact, even for moderate values of n an error probability that is of the order of 2^{-n} is so small that for all practical purposes, probabilistic algorithms are just as good as deterministic algorithms.

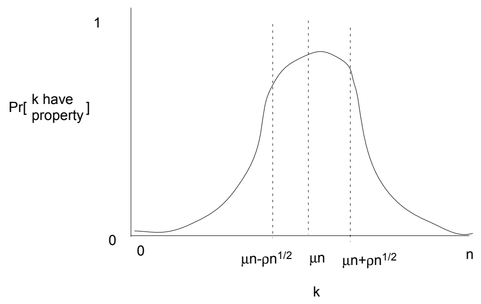
If the original probabilistic algorithm used m coins, then the error reduction procedure we use (run k independent trials and output the majority answer) takes $O(m \cdot k)$ random coins to reduce the error to a value exponentially small in k. It is somewhat surprising that we can in fact do better, and reduce the error to the same level using only $O(m + k)$ random bits (see Section 7.5).

Note 7.13 (The Chernoff Bound)

The Chernoff bound is extensively used (sometimes under different names) in many areas of computer science and other sciences. A typical scenario is the following: there is a universe $\mathcal{U}$ of objects, a fraction μ of them have a certain property, and we wish to estimate μ. For example, in the proof of Theorem 7.10 the universe was the set of 2^m possible coin tosses of some probabilistic algorithm and we wanted to know how many of them cause the algorithm to accept its input. Another example is that $\mathcal{U}$ may be the set of all the citizens of the United States, and we wish to find out how many of them approve of the current president.

A natural approach to compute the fraction μ is to *sample* n members of the universe independently at random, find out the number k of the sample's members that have the property and to estimate that μ is k/n. Of course, it may be quite possible that 10% of the population supports the president, but in a sample of 1000 we will find 101 and not 100 such people, and so we set our goal only to estimate μ up to an *error* of $\pm\epsilon$ for some $\epsilon > 0$. Similarly, even if only 10% of the population have a certain property, we may be extremely unlucky and select only people having it for our sample, and so we allow a small *probability of failure* δ that our estimate will be off by more than ϵ. The natural question is *how many samples do we need to estimate μ up to an error of $\pm\epsilon$ with probability at least $1-\delta$?* The Chernoff bound tells us that (considering μ as a constant) this number is $O(\log(1/\delta)/\epsilon^2)$.

This implies that if we sample n elements, then the probability that the number k having the property is $\rho\sqrt{n}$ far from μn decays *exponentially* with ρ: that is, this probability has the famous "bell curve" shape:



We will use this exponential decay phenomena several times in this book, starting with the proof of Theorem 7.17, showing that **BPP** $\subseteq$ **P**/poly.

7.4.2 Expected running time versus worst-case running time.

When defining **RTIME**$(T(n))$ and **BPTIME**$(T(n))$ we required the machine to halt in $T(n)$ time regardless of its random choices. We could have used *expected* running time instead, as in the definition of **ZPP** (Definition 7.7). It turns out this yields an equivalent definition: we can add a time counter to a PTM M whose expected running time is $T(n)$ and ensure it always halts after at most $100T(n)$ steps. By Markov's inequality (see Lemma A.10), the probability that M runs for more than this time is at most $1/100$. Thus by halting after $100T(n)$ steps, the acceptance probability is changed by at most $1/100$.

7.4.3 Allowing more general random choices than a fair random coin.

One could conceive of real-life computers that have a "coin" that comes up heads with probability ρ that is not $1/2$. We call such a coin a ρ-coin. Indeed it is conceivable that for a random source based upon quantum mechanics, ρ is an irrational number, such as $1/e$. Could such a coin give probabilistic algorithms new power? The following claim shows that it will not.

LEMMA 7.14
A coin with $\Pr[Heads] = \rho$ *can be simulated by a PTM in expected time* $O(1)$ *provided the* i*th bit of* ρ *is computable in poly*(i) *time.*

PROOF: Let the binary expansion of ρ be $0.p_1p_2p_3\ldots$. The PTM generates a sequence of random bits $b_1, b_2, \ldots$, one by one, where b_i is generated at step i. If $b_i < p_i$ then the machine outputs "heads" and stops; if $b_i > p_i$ the machine outputs "tails" and halts; otherwise the machine goes to step $i+1$. Clearly, the machine reaches step $i+1$ iff $b_j = p_j$ for all $j \leq i$, which happens with probability $1/2^i$. Thus the probability of "heads" is $\sum_i p_i \frac{1}{2^i}$, which is exactly ρ. Furthermore, the expected running time is $\sum_i i^c \cdot \frac{1}{2^i}$. For every constant c this infinite sum is upperbounded by another constant (see Exercise 1). ■

Conversely, probabilistic algorithms that only have access to ρ-coins do not have less power than standard probabilistic algorithms:

LEMMA 7.15 (VON-NEUMANN)
A coin with $\Pr[Heads] = 1/2$ *can be simulated by a probabilistic TM with access to a stream of* ρ*-biased coins in expected time* $O(\frac{1}{\rho(1-\rho)})$.

PROOF: We construct a TM M that given the ability to toss ρ-coins, outputs a $1/2$-coin. The machine M tosses pairs of coins until the first time it gets two different results one after the other. If these two results were first "heads" and then "tails", M outputs "heads". If these two results were first "tails" and then "heads", M outputs "tails". For each pair, the probability we get two "heads" is ρ^2, the probability we get two "tails" is $(1-\rho)^2$, the probability we get "head" and then"tails" is $\rho(1-\rho)$, and the probability we get "tails" and then "head" is $(1-\rho)\rho$. We see that the probability we halt and output in each step is $2\rho(1-\rho)$, and that conditioned on this, we do indeed output either "heads" or "tails" with the same probability. Note that we did not need to know ρ to run this simulation. ■

Weak random sources. Physicists (and philosophers) are still not completely certain that randomness exists in the world, and even if it does, it is not clear that our computers have access to an endless stream of independent coins. Conceivably, it may be the case that we only have access to a source of *imperfect* randomness, that although unpredictable, does not consist of independent coins. As we will see in Chapter 16, we do know how to simulate probabilistic algorithms designed for perfect independent 1/2-coins even using such a weak random source.

7.5 Randomness efficient error reduction.

In Section 7.4.1 we saw how we can reduce error of probabilistic algorithms by running them several time using independent random bits each time. Ideally, one would like to be frugal with using randomness, because good quality random number generators tend to be slower than the rest of the computer. Surprisingly, the error reduction can be done just as effectively without using truly independent runs, and "recycling" the random bits. Now we outline this idea; a much more general theory will be later presented in Chapter 16.

The main tool we use is *expander* graphs. Expander graphs have played a crucial role in numerous computer science applications, including routing networks, error correcting codes, hardness of approximation and the PCP theorem, derandomization, and more. Expanders can be defined in several roughly equivalent ways. One is that these are graphs where every set of vertices has a very large boundary. That is, for every subset S of vertices, the number of S's neighbors outside S is (up to a constant factor) roughly equal to the number of vertices inside S. (Of course this condition cannot hold if S is too big and already contains almost all of the vertices in the graph.) For example, the n by n grid (where a vertex is a pair (i, j) and is connected to the four neighbors $(i \pm 1, j \pm 1)$) is *not* an expander, as any k by k square (which is a set of size k^2) in this graph only has a boundary of size $O(k)$ (see Figure 7.1).

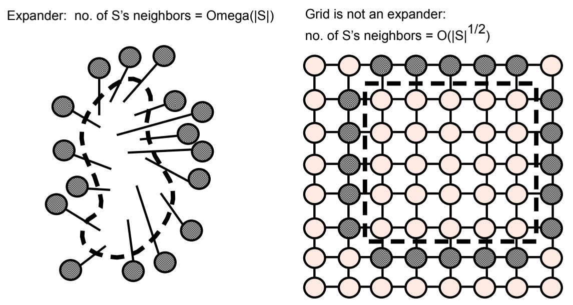

Expander: no. of S's neighbors = Omega(|S|)
Grid is not an expander:
no. of S's neighbors = O(|S|^{1/2})

Figure 7.1: In a *combinatorial expander*, every subset S of the vertices that is not too big has at least $\Omega(|S|)$ neighbors outside the set. The grid (and every other planar graph) is not a combinatorial expander as a $k \times k$ square in the grid has only $O(k)$ neighbors outside it.

We will not precisely define expanders now (but see Section 7.B at the end of the chapter). However, an *expander graph family* is a sequence of graphs $\{G_N\}_{N\in\mathbb{N}}$ such for every N, G_N is an N-vertex D-degree graph for some constant D. Deep mathematics (and more recently, simpler mathematics) has been used to construct expander graphs. These constructions yield algorithms that, given the binary representation of N and an index of a node in G_N, can produce the indices of the D neighbors of this node in poly($\log N$) time.

We illustrate the error reduction procedure by showing how we transform an **RP** algorithm that outputs the right answer with probability 1/2into an algorithm that outputs the right answer with probability $1 - 2^{-\Omega(k)}$. The idea is simple: let x be an input, and suppose we have an algorithm M using m coins such that if $x \in L$ then $\Pr_{r \in_R \{0,1\}^m}[M(x,r) = 1] \geq 1/2$ and if $x \notin L$ then $M(x,r) = 0$ for every r. Let $N = 2^m$ and let G_N be an N-vertex expander family. We use m coins to select a random vertex v from G_N, and then use $\log Dk$ coins to take a $k-1$-*step random walk* from v on G_N. That is, at each step we choose a random number i in $[D]$ and move from the current vertex to its i^{th} neighbor. Let $v_1, \ldots, v_k$ be the vertices we encounter along this walk (where $v_1 = v$). We can treat these vertices as elements of $\{0,1\}^m$ and run the machine M on input x with all of these coins. If even one of these runs outputs 1, then output 1. Otherwise, output 0. It can be shown that if less than half of the r's cause M to output 0, then the probability that the walk is fully contained in these "bad" r's is exponentially small in k.

We see that what we need to prove is the following theorem:

Theorem 7.16
Let G be an expander graph of N vertices and B a subset of G's vertices of size at most βN, where $\beta < 1$. Then, the probability that a k-vertex random walk is fully contained in B is at most ρ^k, where $\rho < 1$ is a constant depending only on β (and independent of k).

Theorem 7.16 makes intuitive sense, as in an expander graph a constant fraction of the edges adjacent to vertices of B will have the other vertex in B's complement, and so it seems that at each step we will have a constant probability to leave B. However, its precise formulation and analysis takes some care, and is done at the end of the chapter in Section 7.B.

Intuitively, We postpone the full description of the error reduction procedure and its analysis to Section 7.B.

7.6 BPP $\subseteq$ P/poly

Now we show that all **BPP** languages have polynomial sized circuits. Together with Theorem **??** this implies that if 3SAT $\in$ **BPP** then **PH** $= \mathbf{\Sigma}_2^p$.

Theorem 7.17 (Adleman)
BPP $\subseteq$ **P**/poly.

Proof: Suppose $L \in$ **BPP**, then by the alternative definition of **BPP** and the error reduction procedure of Theorem 7.10, there exists a TM M that on inputs of size n uses m random bits and satisfies

$$x \in L \Rightarrow \mathbf{Pr}_r\left[M(x,r) \text{ accepts }\right] \geq 1 - 2^{-(n+2)}$$
$$x \notin L \Rightarrow \mathbf{Pr}_r\left[M(x,r) \text{ accepts }\right] \leq 2^{-(n+2)}$$

(Such a machine exists by the error reduction arguments mentioned earlier.)

Say that an $r \in \{0,1\}^m$ is *bad* for an input $x \in \{0,1\}^n$ if $M(x,r)$ is an incorrect answer, otherwise we say its *good* for x. For every x, at most $2 \cdot 2^m/2^{(n+2)}$ values of r are bad for x. Adding over all $x \in \{0,1\}^n$, we conclude that at most $2^n \times 2^m/2^{(n+1)} = 2^m/2$ strings r are bad for *some* x. In other words, at least $2^m - 2^m/2$ choices of r are good for *every* $x \in \{0,1\}^n$. Given a string r_0 that is good for every $x \in \{0,1\}^n$, we can hardwire it to obtain a circuit C (of size at most quadratic in the running time of M) that on input x outputs $M(x,r_0)$. The circuit C will satisfy $C(x) = L(x)$ for every $x \in \{0,1\}^n$. ■

7.7 BPP is in PH

At first glance, **BPP** seems to have nothing to do with the polynomial hierarchy, so the next theorem is somewhat surprising.

THEOREM 7.18 (SIPSER-GÁCS)
$\mathbf{BPP} \subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$

PROOF: It is enough to prove that $\mathbf{BPP} \subseteq \mathbf{\Sigma}_2^p$ because **BPP** is closed under complementation (i.e., $\mathbf{BPP} = \mathbf{coBPP}$).

Suppose $L \in \mathbf{BPP}$. Then by the alternative definition of **BPP** and the error reduction procedure of Theorem 7.10 there exists a polynomial-time deterministic TM M for L that on inputs of length n uses $m = \text{poly}(n)$ random bits and satisfies

$$x \in L \Rightarrow \mathbf{Pr}_r\left[M(x,r) \text{ accepts }\right] \geq 1 - 2^{-n}$$
$$x \notin L \Rightarrow \mathbf{Pr}_r\left[M(x,r) \text{ accepts }\right] \leq 2^{-n}$$

For $x \in \{0,1\}^n$, let S_x denote the set of r's for which M accepts the input pair (x,r). Then either $|S_x| \geq (1-2^{-n})2^m$ or $|S_x| \leq 2^{-n}2^m$, depending on whether or not $x \in L$. We will show how to check, using two alternations, which of the two cases is true.

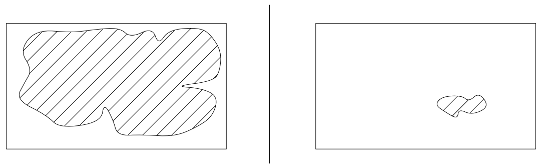


Figure 7.2: There are only two possible sizes for the set of r's such that $M(x,r)$ =Accept: either this set is almost all of $\{0,1\}^m$ or a tiny fraction of $\{0,1\}^m$. In the former case, a few random "shifts" of this set are quite likely to cover all of $\{0,1\}^m$. In the latter case the set's size is so small that a few shifts cannot cover $\{0,1\}^m$

For $k = \frac{m}{n} + 1$, let $U = \{u_1, \ldots, u_k\}$ be a set of k strings in $\{0,1\}^m$. We define G_U to be a graph with vertex set $\{0,1\}^m$ and edges (r,s) for every r,s such that $r = s + u_i$ for some $i \in [k]$

(where + denotes vector addition modulo 2, or equivalently, bitwise XOR). Note that the degree of G_U is k. For a set $S \subseteq \{0,1\}^m$, define $\Gamma_U(S)$ to be all the neighbors of S in the graph G_U. That is, $r \in \Gamma_U(S)$ if there is an $s \in S$ and $i \in [k]$ such that $r = s + u_i$.

Claim 1: For every set $S \subseteq \{0,1\}^m$ with $|S| \leq 2^{m-n}$ and every set U of size k, it holds that $\Gamma_U(S) \neq \{0,1\}^m$. Indeed, since Γ_U has degree k, it holds that $|\Gamma_U(S)| \leq k|S| < 2^m$.

Claim 2: For every set $S \subseteq \{0,1\}^m$ with $|S| \geq (1-2^{-n})2^m$ there exists a set U of size k such that $\Gamma_U(S) = \{0,1\}^m$. We show this by the probabilistic method, by proving that for every S, if we choose U at random by taking k random strings $u_1, \ldots, u_k$, then $\Pr[\Gamma_U(S) = \{0,1\}^m] > 0$. Indeed, for $r \in \{0,1\}^m$, let B_r denote the "bad event" that r is not in $\Gamma_U(S)$. Then, $B_r = \cap_{i\in[k]} B_r^i$ where B_r^i is the event that $r \notin S + u_i$, or equivalently, that $r + u_i \notin S$ (using the fact that modulo 2, $a + b = c \Leftrightarrow a = c + b$). Yet, $r + u_i$ is a uniform element in $\{0,1\}^m$, and so it will be in S with probability at least $1 - 2^{-n}$. Since $B_r^1, \ldots, B_r^k$ are independent, the probability that B_r happens is at most $(1-2^{-n})^k < 2^{-m}$. By the union bound, the probability that $\Gamma_U(S) \neq \{0,1\}^m$ is bounded by $\sum_{r\in\{0,1\}^m} \Pr[B_r] < 1$.

Together Claims 1 and 2 show $x \in L$ if and only if the following statement is true

$$\exists u_1, \ldots, u_k \in \{0,1\}^m \ \forall r \in \{0,1\}^m \ \bigvee_{i=1}^{k} M(x, r \oplus u_i) \text{accepts}$$

thus showing $L \in \mathbf{\Sigma_2}$. ■

7.8 State of our knowledge about BPP

We know that $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{P}/poly$, and furthermore, that $\mathbf{BPP} \subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$ and so if $\mathbf{NP} = p$ then $\mathbf{BPP} = \mathbf{P}$. As mentioned above, there are complexity-theoretic reasons to strongly believe that $\mathbf{BPP} \subseteq \mathbf{DTIME}(2^{\epsilon})$ for every $\epsilon > 0$, and in fact to reasonably suspect that $\mathbf{BPP} = \mathbf{P}$ (see Chapters 16 and 17). However, currently we are not even able to rule out that $\mathbf{BPP} = \mathbf{NEXP}$!

Complete problems for BPP?

Though a very natural class, **BPP** behaves differently in some ways from other classes we have seen. For example, we know of no complete languages for it (under deterministic polynomial time reductions). One reason for this difficulty is that the defining property of **BPTIME** machines is *semantic*, namely, that for *every* string they either accept with probability at least 2/3 or reject with probability at least 1/3. Given the description of a Turing machine M, testing whether it has this property is undecidable. By contrast, the defining property of an NDTM is *syntactic:* given a string it is easy to determine if it is a valid encoding of an NDTM. Completeness seems easier to define for syntactically defined classes than for semantically defined ones. For example, consider the following natural attempt at a **BPP**-complete language: $L = \{\langle M, x\rangle : \Pr[M(x) = 1] \geq 2/3\}$. This language is indeed **BPP**-hard but is not known to be in **BPP**. In fact, it is not in any level of the polynomial hierarchy unless the hierarchy collapses. We note that if, as believed, $\mathbf{BPP} = \mathbf{P}$, then **BPP** does have a complete problem. (One can sidestep some of the above issues by using *promise* problems instead of languages, but we will not explore this.)

Does BPTIME have a hierarchy theorem?

Is $\mathbf{BPTIME}(n^c)$ contained in $\mathbf{BPTIME}(n)$ for some $c > 1$? One would imagine not, and this seems as the kind of result we should be able to prove using the tools of Chapter 3. However currently we are even unable to show that $\mathbf{BPTIME}(n^{\log^2 n})$ (say) is not in $\mathbf{BPTIME}(n)$. The standard diagonalization techniques fail, for similar reasons as the ones above. However, recently there has been some progress on obtaining hierarchy theorem for some closely related classes (see notes).

7.9 Randomized reductions

Since we have defined randomized algorithms, it also makes sense to define a notion of randomized reduction between two languages. This proves useful in some complexity settings (e.g., see Chapters 9 and 8).

Definition 7.19
Language A reduces to language B under a randomized polynomial time reduction, denoted $A \le_r B$, if there is a probabilistic TM M such that for every $x \in \{0,1\}^*$, $\Pr[B(M(x)) = A(x)] \ge 2/3$.

We note that if $A \le_r B$ and $B \in \mathbf{BPP}$ then $A \in \mathbf{BPP}$. This alerts us to the possibility that we could have defined **NP**-completeness using randomized reductions instead of deterministic reductions, since arguably **BPP** is as good as **P** as a formalization of the notion of efficient computation. Recall that the Cook-Levin theorem shows that **NP** may be defined as the set $\{L : L \le_p \mathsf{3SAT}\}$. The following definition is analogous.

Definition 7.20 ($\mathbf{BP \cdot NP}$)
$\mathbf{BP \cdot NP} = \{L : L \le_r \mathsf{3SAT}\}$.

We explore the properties of $\mathbf{BP \cdot NP}$ in the exercises, including whether or not $\overline{\mathsf{3SAT}} \in \mathbf{BP \cdot NP}$.

One interesting application of randomized reductions will be shown in Chapter 9, where we present a (variant of a) randomized reduction from 3SAT to the solving special case of 3SAT where we are guaranteed that the formula is either unsatisfiable or has a *single unique* satisfying assignment.

7.10 Randomized space-bounded computation

A PTM is said to work in space $S(n)$ if every branch requires space $O(S(n))$ on inputs of size n and terminates in $2^{O(S(n))}$ time. Recall that the machine has a read-only input tape, and the work space only cell refers only to its read/write work tapes. As a PTM it has two transition functions that are applied with equal probability. The most interesting case is when the work tape has $O(\log n)$ size. The classes **BPL** and **RL** are the two-sided error and one-sided error probabilistic analogs of the class **L** defined in Chapter 4.

Definition 7.21 ([)
The classes **BPL** and **RL**] A language L is in **BPL** if there is an $O(\log n)$-space probabilistic TM M such that $\Pr[M(x) = L(x)] \geq 2/3$.
A language L is in **RL** if there is an $O(\log n)$-space probabilistic TM M such that if $x \in L$ then $\Pr[M(x) = 1] \geq 2/3$ and if $x \notin L$ then $\Pr[M(x) = 1] = 0$.

The reader can verify that the error reduction procedure described in Chapter 7 can be implemented with only logarithmic space overhead, and hence also in these definitions the choice of the precise constant is not significant. We note that $\mathbf{RL} \subseteq \mathbf{NL}$, and thus $\mathbf{RL} \subseteq \mathbf{P}$. The exercises ask you to show that $\mathbf{BPL} \subseteq \mathbf{P}$ as well.

One famous **RL**-algorithm is the algorithm to solve UPATH: the restriction of the **NL**-complete PATH problem (see Chapter 4) to undirected graphs. That is, given an n-vertex undirected graph G and two vertices s and t, determine whether s is connected to t in G.

Theorem 7.22 ([**?**])
UPATH $\in$ **RL**.

The algorithm for UPATH is actually very simple: take a random walk of length n^3 starting from s. That is, initialize the variable v to the vertex s and in each step choose a random neighbor u of v, and set $v \leftarrow u$. Accept iff the walk reaches t within n^3 steps. Clearly, if s is not connected to t then the algorithm will never accept. It can be shown that if s is connected to t then the expected number of steps it takes for a walk from s to hit t is at most $\frac{4}{27}n^3$ and hence our algorithm will accept with probability at least $\frac{3}{4}$. We defer the analysis of this algorithm to the end of the chapter at Section 7.A, where we will prove that a somewhat larger walk suffices to hit t with good probability (see also Exercise 9).

In Chapter 16 we show a recent *deterministic* logspace algorithm for the same problem. It is known that **BPL** (and hence also **RL**) is contained in $\mathbf{SPACE}(\log^{3/2} n)$. In Chapter 16 we will see a somewhat weaker result: a simulation of **BPL** in $\log^2 n$ space and polynomial time.

What have we learned?

- The class **BPP** consists of languages that can be solved by a probabilistic polynomial-time algorithm. The probability is only over the algorithm's coins and not the choice of input. It is arguably a better formalization of efficient computation than **P**.
- **RP**, **coRP** and **ZPP** are subclasses of **BPP** corresponding to probabilistic algorithms with one-sided and "zero-sided" error.
- Using repetition, we can considerably amplify the success probability of probabilistic algorithms.
- We only know that $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{EXP}$, but we suspect that $\mathbf{BPP} = \mathbf{P}$.
- **BPP** is a subset of both **P**/poly and **PH**. In particular, the latter implies that if $\mathbf{NP} = \mathbf{P}$ then $\mathbf{BPP} = \mathbf{P}$.
- Randomness is used in complexity theory in many contexts beyond **BPP**. Two examples are randomized reductions and randomized logspace algorithms, but we will see many more later.

Chapter notes and history

Early researchers realized the power of randomization since their computations —e.g., for design of nuclear weapons— used probabilistic tools such as Monte Carlo simulations. Papers by von Neumann [**?**] and de Leeuw et al. [**?**] describe probabilistic Turing machines. The definitions of **BPP**, **RP** and **ZPP** are from Gill [**?**]. (In an earlier conference paper [**?**], Gill studies similar issues but seems to miss the point that a practical algorithm for deciding a language must feature a *gap* between the acceptance probability in the two cases.)

The algorithm used to show PRIMES is in **coRP** is due to Solovay and Strassen [**?**]. Another primality test from the same era is due to Rabin [**?**]. Over the years, better tests were proposed. In a recent breakthrough, Agrawal, Kayal and Saxena finally proved that $\mathsf{PRIMES} \in \mathbf{P}$. Both the probabilistic and deterministic primality testing algorithms are described in Shoup's book [**?**].

Lovász's randomized **NC** algorithm [**?**] for deciding the *existence* of perfect matchings is unsatisfying in the sense that when it outputs "Accept," it gives no clue how to find a matching! Subsequent probabilistic **NC** algorithms can find a perfect matching as well; see [**?**, **?**].

$\mathbf{BPP} \subseteq \mathbf{P}$/poly is from Adelman [**?**]. $\mathbf{BPP} \subseteq \mathbf{PH}$ is due to Sipser [**?**], and the stronger form $\mathbf{BPP} \subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$ is due to P. Gács. Recent work [] shows that **BPP** is contained in classes that are seemingly weaker than $\mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$.

Even though a hierarchy theorem for **BPP** seems beyond our reach, there has been some success in showing hierarchy theorems for the seemingly related class **BPP**/1 (i.e., **BPP** with a single bit of nonuniform advice) [**?**, **?**, **?**].

Readers interested in randomized algorithms are referred to the books by Mitzenmacher and Upfal [?] and Motwani and Raghavan [?].

STILL A LOT MISSING

Expanders were well-studied for a variety of reasons in the 1970s but their application to pseudorandomness was first described by Ajtai, Komlos, and Szemeredi [?]. Then Cohen-Wigderson [?] and Impagliazzo-Zuckerman (1989) showed how to use them to "recycle" random bits as described in Section 7.B.3. The upcoming book by Hoory, Linial and Wigderson (draft available from their web pages) provides an excellent introduction to expander graphs and their applications.

The explicit construction of expanders is due to Reingold, Vadhan and Wigderson [?], although we chose to present it using the replacement product as opposed to the closely related zig-zag product used there. The deterministic logspace algorithm for undirected connectivity is due to Reingold [?].

Exercises

§1 Show that for every $c > 0$, the following infinite sum is finite:

$$\sum_{i \geq 1} \frac{i^c}{2^i}.$$

§2 Show, given input the numbers a, n, p (in binary representation), how to compute $a^n (\mathrm{mod} p)$ in polynomial time.

Hint: use the binary representation of n and repeated squaring.

§3 Let us study to what extent Claim **??** truly needs the assumption that ρ is efficiently computable. Describe a real number ρ such that given a random coin that comes up "Heads" with probability ρ, a Turing machine can decide an undecidable language in polynomial time.

Hint: think of the real number ρ as an advice string. How can its bits be recovered?

§4 Show that $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$.

§5 A nondeterministic circuit has two inputs x, y. We say that it accepts x iff there exists y such that $C(x, y) = 1$. The size of the circuit is measured as a function of $|x|$. Let $\mathbf{NP}/poly$ be the languages that are decided by polynomial size nondeterministic circuits. Show that $BP \cdot \mathbf{NP} \subseteq \mathbf{NP}/poly$.

§6 Show using ideas similar to the Karp-Lipton theorem that if $\overline{\mathsf{3SAT}} \in BP \cdot \mathbf{NP}$ then $\mathbf{PH}$ collapses to $\mathbf{\Sigma}_3^p$. (Combined with above, this shows it is unlikely that $\mathsf{3SAT} \leq_r \overline{\mathsf{3SAT}}$.)

§7 Show that $\mathbf{BPL} \subseteq \mathbf{P}$

Hint: try to compute the probability that the machine ends up in the accept configuration using either dynamic programming or matrix multiplication.

§8 Show that the random walk idea for solving connectivity does not work for directed graphs. In other words, describe a directed graph on n vertices and a starting point s such that the expected time to reach t is $\Omega(2^n)$ even though there is a directed path from s to t.

§9 Let G be an n vertex graph where all vertices have the same degree.

(a) We say that a distribution $\mathbf{p}$ over the vertices of G (where $\mathbf{p}_i$ denotes the probability that vertex i is picked by $\mathbf{p}$) is *stable* if when we choose a vertex i according to $\mathbf{p}$ and take a random step from i (i.e., move to a random neighbor j or i) then the resulting distribution is $\mathbf{p}$. Prove that the uniform distribution on G's vertices is stable.

(b) For $\mathbf{p}$ be a distribution over the vertices of G, let $\Delta(\mathbf{p}) = \max_i\{\mathbf{p}_i - 1/n\}$. For every k, denote by $\mathbf{p}^k$ the distribution obtained by choosing a vertex i at random from $\mathbf{p}$ and taking k random steps on G. Prove that if G is connected then there exists k such that $\Delta(\mathbf{p}^k) \le (1 - n^{-10n})\Delta(\mathbf{p})$. Conclude that

i. The uniform distribution is the only stable distribution for G.

ii. For every vertices u, v of G, if we take a sufficiently long random walk starting from u, then with high probability the fraction of times we hit the vertex v is roughly $1/n$. That is, for every $\epsilon > 0$, there exists k such that the k-step random walk from u hits v between $(1-\epsilon)k/n$ and $(1+\epsilon)k/n$ times with probability at least $1-\epsilon$.

(c) For a vertex u in G, denote by E_u the expected number of steps it takes for a random walk starting from u to reach back u. Show that $E_u \le 10n^2$.

Hint: consider the infinite random walk starting from u. If $E_u > K$ then by standard tail bounds, u appears in less than a $2/K$ fraction of the places in this walk.

(d) For every two vertices u, v denote by $E_{u,v}$ the expected number of steps it takes for a random walk starting from u to reach v. Show that if u and v are connected by a path of length at most k then $E_{u,v} \le 100kn^2$. Conclude that for every s and t that are connected in a graph G, the probability that an $1000n^3$ random walk from s does not hit t is at most $1/10$.

Hint: Start with the case $k = 1$ (i.e., u and v are connected by an edge), the case of $k > 1$ can be reduced to this using linearity of expectation. Note that the expectation of a random variable X over $\mathbb{N}$ is equal to $\sum_{m\in\mathbb{N}} \Pr[X \ge m]$ and so it suffices to show that the probability that an ℓn^2-step random walk from u does not hit v decays exponentially with ℓ.

(e) Let G be an n-vertex graph that is not necessarily regular (i.e., each vertex may have different degree). Let G' be the graph obtained by adding a sufficient number of parallel self-loops to each vertex to make G regular. Prove that if a k-step random walk in G' from a vertex s hits a vertex t with probability at least 0.9, then a $10n^2k$-step random walk from s will hit t with probability at least $1/2$.

The following exercises are based on Sections 7.A and 7.B.

§10 Let A be a symmetric stochastic matrix: $A = A^\dagger$ and every row and column of A has non-negative entries summing up to one. Prove that $\|A\| \leq 1$.

Hint: first show that $\|A\|$ is at most say n^2. Then, prove that for every $k \geq 1$, A^k is also stochastic and $\|A^{2k}\mathbf{v}\|_2 \geq \|A^k\mathbf{v}\|_2^2$ using the equality $\langle \mathbf{w}, B\mathbf{z}\rangle = \langle B^\dagger \mathbf{w}, \mathbf{z}\rangle$ and the inequality $\langle \mathbf{w}, \mathbf{z}\rangle \leq \|\mathbf{w}\|_2 \|\mathbf{z}\|_2$.

§11 Let A, B be two symmetric stochastic matrices. Prove that $\lambda(A+B) \leq \lambda(A) + \lambda(B)$.

§12 Let a n, d random graph be an n-vertex graph chosen as follows: choose d random permutations $\pi_1, ldots, \pi_d$ from $[n]$ to $[n]$. Let the the graph G contains an edge (u, v) for every pair u, v such that $v = \pi_i(u)$ for some $1 \leq i \leq d$. Prove that a random n, d graph is an $(n, 2d, \frac{2}{3}d)$ combinatorial expander with probability $1 - o(1)$ (i.e., tending to one with n).

Hint: for every set $S \subseteq n$ with $|S| \leq n/2$ and set $T \subseteq [n]$ with $|T| \leq (1 + \frac{2}{3}d)|S|$, try to bound the probability that $\pi_i(S) \subseteq T$ for every i.

The following two section assume some knowledge of elementary linear algebra (vector spaces and Hilbert spaces); see Appendix A for a quick review.

7.A Random walks and eigenvalues

In this section we study random walks on (undirected regular) graphs, introducing several important notions such as the spectral gap of a graph's adjacency matrix. As a corollary we obtain the proof of correctness for the random-walk space-efficient algorithm for UPATH of Theorem 7.22. We will see that we can use elementary linear algebra to relate parameters of the graph's adjacency matrix to the behavior of the random walk on that graph.

Remark 7.23
In this section, we restrict ourselves to *regular* graphs, in which every vertex have the same degree, although the definitions and results can be suitably generalized to general (non-regular) graphs.

7.A.1 Distributions as vectors and the parameter $\lambda(G)$.

Let G be a d-regular n-vertex graph. Let $\mathbf{p}$ be some probability distribution over the vertices of G. We can think of $\mathbf{p}$ as a (column) vector in $\mathbb{R}^n$ where $\mathbf{p}_i$ is the probability that vertex i is obtained by the distribution. Note that the L_1-norm of $\mathbf{p}$ (see Note 7.24), defined as $|\mathbf{p}|_1 = \sum_{i=1}^n |\mathbf{p}_i|$, is equal to 1. (In this case the absolute value is redundant since $\mathbf{p}_i$ is always between 0 and 1.)

Now let $\mathbf{q}$ represent the distribution of the following random variable: choose a vertex i in G according to $\mathbf{p}$, then take a random neighbor of i in G. We can compute $\mathbf{q}$ as a function of $\mathbf{p}$: the probability $\mathbf{q}_j$ that j is chosen is equal to the sum over all j's neighbors i of the probability $\mathbf{p}_i$ that i is chosen times $1/d$ (where $1/d$ is the probability that, conditioned on i being chosen, the walk moves to $\mathbf{q}$). Thus $\mathbf{q} = A\mathbf{p}$, where $A = A(G)$ which is the *normalized adjacency matrix* of G. That is, for every two vertices i, j, $A_{i,j}$ is equal to the number of edges between i and j divided by d. Note that A is a symmetric matrix,[3] where each entry is between 0 and 1, and the sum of entries in each row and column is exactly one (such a matrix is called a symmetric *stochastic* matrix).

Let $\{\mathbf{e}^i\}_{i=1}^n$ be the *standard basis* of $\mathbb{R}^n$ (i.e. $\mathbf{e}^i$ has 1 in the i^{th} coordinate and zero everywhere else). Then, $A^T\mathbf{e}^s$ represents the distribution X_T of taking a T-step random walk from the vertex s. This already suggests that considering the adjacency matrix of a graph G could be very useful in analyzing random walks on G.

Definition 7.25 (The parameter $\lambda(G)$.)
Denote by $\mathbf{1}$ the vector $(1/n, 1/n, \ldots, 1/n)$ corresponding to the uniform distribution. Denote by $\mathbf{1}^\perp$ the set of vectors perpendicular to $\mathbf{1}$ (i.e., $\mathbf{v} \in \mathbf{1}^\perp$ if $\langle \mathbf{v}, \mathbf{1}\rangle = (1/n)\sum_i \mathbf{v}_i = 0$).
The parameter $\lambda(A)$, denoted also as $\lambda(G)$, is the maximum value of $\|A\mathbf{v}\|_2$ over all vectors $\mathbf{v} \in \mathbf{1}^\perp$ with $\|\mathbf{v}\|_2 = 1$.

[3] A matrix A is *symmetric* if $A = A^\dagger$, where $A^\dagger$ denotes the *transpose* of A. That is, $(A^\dagger)_{i,j} = A_{j,i}$ for every i, j.

Note 7.24 (L_p Norms)
A *norm* is a function mapping a vector $\mathbf{v}$ into a real number $\|\mathbf{v}\|$ satisfying **(1)** $\|\mathbf{v}\| \geq 0$ with $\|\mathbf{v}\| = 0$ if and only $\mathbf{v}$ is the all zero vector, **(2)** $\|\alpha\mathbf{v}\| = |\alpha| \cdot \|\mathbf{v}\|$ for every $\alpha \in \mathbb{R}$, and **(3)** $\|\mathbf{v}+\mathbf{u}\| \leq \|\mathbf{v}\| + \|\mathbf{u}\|$ for every vector $\mathbf{u}$. The third inequality implies that for every norm, if we define the *distance* between two vectors $\mathbf{u}$,$\mathbf{v}$ as $\|\mathbf{u}-\mathbf{v}\|$ then this notion of distance satisfies the triangle inequality.

For every $\mathbf{v} \in \mathbb{R}^n$ and number $p \geq 1$, the L_p *norm* of $\mathbf{v}$, denoted $\|\mathbf{v}\|_p$, is equal to $\left(\sum_{i=1}^n |\mathbf{v}_i|^p\right)^{1/p}$. One particularly interesting case is $p = 2$, the so-called *Euclidean norm*, in which $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n \mathbf{v}_i^2} = \sqrt{\langle \mathbf{v}, \mathbf{v}\rangle}$. Another interesting case is $p = 1$, where we use the single bar notation and denote $|\mathbf{v}|_1 = \sum_{i=1}^n |\mathbf{v}_i|$. Another case is $p = \infty$, where we denote $\|\mathbf{v}\|_\infty = \lim_{p\to\infty} \|\mathbf{v}\|_p = \max_{i\in[n]} |\mathbf{v}_i|$.

The *Hölder inequality* says that for every p, q with $\frac{1}{p} + \frac{1}{q} = 1$, $\|\mathbf{u}\|_p \|\mathbf{v}\|_q \geq \sum_{i=1}^n |\mathbf{u}_i\mathbf{v}_i|$. To prove it, note that by simple scaling, it suffices to consider norm one vectors, and so it enough to show that if $\|\mathbf{u}\|_p = \|\mathbf{v}\|_q = 1$ then $\sum_{i=1}^n |\mathbf{u}_i||\mathbf{v}_i| \leq 1$. But $\sum_{i=1}^n |\mathbf{u}_i||\mathbf{v}_i| = \sum_{i=1}^n |\mathbf{u}_i|^{p(1/p)}|\mathbf{v}_i|^{q(1/q)} \leq \sum_{i=1}^n \frac{1}{p}|\mathbf{u}_i|^p + \frac{1}{q}|\mathbf{v}_i|^q = \frac{1}{p} + \frac{1}{q} = 1$, where the last inequality uses the fact that for every $a, b > 0$ and $\alpha \in [0,1]$, $a^\alpha b^{1-\alpha} \leq \alpha a + (1-\alpha)b$. This fact is due to the log function being concave— having negative second derivative, implying that $\alpha \log a + (1-\alpha)\log b \leq \log(\alpha a + (1-\alpha)b)$.
Setting $p = 1$ and $q = \infty$, the Hölder inequality implies that

$$\|\mathbf{v}\|_2 \leq |\mathbf{v}|_1 \|\mathbf{v}\|_\infty$$

Setting $p = q = 2$, the Hölder inequality becomes the *Cauchy-Schwartz Inequality* stating that $\sum_{i=1}^n |\mathbf{u}_i\mathbf{v}_i| \leq \|\mathbf{u}\|_2\|\mathbf{v}\|_2$. Setting $\mathbf{u} = (1/\sqrt{n}, 1/\sqrt{n}, \ldots, 1/\sqrt{n})$, we get that

$$|\mathbf{v}|_1/\sqrt{n} = \sum_{i=1}^n \tfrac{1}{\sqrt{n}}|\mathbf{v}_i| \leq \|\mathbf{v}\|_2$$

Remark 7.26
The value $\lambda(G)$ is often called the *second largest eigenvalue* of G. The reason is that since A is a symmetric matrix, we can find an orthogonal basis of eigenvectors $\mathbf{v}^1, \dots, \mathbf{v}^n$ with corresponding eigenvalues $\lambda_1, \dots, \lambda_n$ which we can sort to ensure $|\lambda_1| \geq |\lambda_2| \dots \geq |\lambda_n|$. Note that $A\mathbf{1} = \mathbf{1}$. Indeed, for every i, $(A\mathbf{1})_i$ is equal to the inner product of the i^{th} row of A and the vector $\mathbf{1}$ which (since the sum of entries in the row is one) is equal to $1/n$. Thus, $\mathbf{1}$ is an *eigenvector* of A with the corresponding eigenvalue equal to 1. One can show that a symmetric stochastic matrix has all eigenvalues with absolute value at most 1 (see Exercise 10) and hence we can assume $\lambda_1 = 1$ and $\mathbf{v}^1 = \mathbf{1}$. Also, because $\mathbf{1}^\perp = \mathrm{Span}\{\mathbf{v}^2, \dots, \mathbf{v}^n\}$, the value λ above will be maximized by (the normalized version of) $\mathbf{v}^2$, and hence $\lambda(G) = |\lambda_2|$. The quantity $1 - \lambda(G)$ is called the *spectral gap* of the graph. We note that some texts use *un-normalized* adjacency matrices, in which case $\lambda(G)$ is a number between 0 and d and the spectral gap is defined to be $d - \lambda(G)$.

One reason that $\lambda(G)$ is an important parameter is the following lemma:

Lemma 7.27
For every regular n vertex graph $G = (V, E)$ let $\mathbf{p}$ be any probability distribution over V, then

$$\|A^T\mathbf{p} - \mathbf{1}\|_2 \leq \lambda^T$$

Proof: By the definition of $\lambda(G)$, $\|A\mathbf{v}\|_2 \leq \lambda\|\mathbf{v}\|_2$ for every $\mathbf{v} \perp \mathbf{1}$. Note that if $\mathbf{v} \perp \mathbf{1}$ then $A\mathbf{v} \perp \mathbf{1}$ since $\langle \mathbf{1}, A\mathbf{v}\rangle = \langle A^\dagger\mathbf{1}, \mathbf{v}\rangle = \langle \mathbf{1}, \mathbf{v}\rangle = 0$ (as $A = A^\dagger$ and $A\mathbf{1} = \mathbf{1}$). Thus A maps the space $\mathbf{1}^\perp$ to itself and since it shrinks any member of this space by at least λ, $\lambda(A^T) \leq \lambda(A)^T$. (In fact, using the eigenvalue definition of λ, it can be shown that $\lambda(A^T) = \lambda(A)$.)

Let $\mathbf{p}$ be some vector. We can break $\mathbf{p}$ into its components in the spaces parallel and orthogonal to $\mathbf{1}$ and express it as $\mathbf{p} = \alpha\mathbf{1} + \mathbf{p}'$ where $\mathbf{p}' \perp \mathbf{1}$ and α is some number. If $\mathbf{p}$ is a probability distribution then $\alpha = 1$ since the sum of coordinates in $\mathbf{p}'$ is zero. Therefore,

$$A^T\mathbf{p} = A^T(\mathbf{1} + \mathbf{p}') = \mathbf{1} + A^T\mathbf{p}'$$

Since $\mathbf{1}$ and $\mathbf{p}'$ are orthogonal, $\|\mathbf{p}\|_2^2 = \|\mathbf{1}\|_2^2 + \|\mathbf{p}'\|_2^2$ and in particular $\|\mathbf{p}'\|_2 \leq \|\mathbf{p}\|_2$. Since $\mathbf{p}$ is a probability vector, $\|\mathbf{p}\|_2 \leq |\mathbf{p}|_1 \cdot 1 \leq 1$ (see Note 7.24). Hence $\|\mathbf{p}'\|_2 \leq 1$ and

$$\|A^T\mathbf{p} - \mathbf{1}\|_2 = \|A^T\mathbf{p}'\|_2 \leq \lambda^T$$

■

It turns out that every connected graph has a noticeable spectral gap:

Lemma 7.28
For every d-regular connected G with self-loops at each vertex, $\lambda(G) \leq 1 - \frac{1}{8dn^3}$.

Proof: Let $\mathbf{u} \perp \mathbf{1}$ be a unit vector and let $\mathbf{v} = A\mathbf{u}$. We'll show that $1 - \|\mathbf{v}\|_2^2 \geq \frac{1}{d4n^3}$ which implies $\|\mathbf{v}\|_2^2 \leq 1 - \frac{1}{d4n^3}$ and hence $\|\mathbf{v}\|_2 \leq 1 - \frac{1}{d8n^3}$.

Since $\|\mathbf{u}\|_2 = 1$, $1 - \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2$. We claim that this is equal to $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$ where i, j range from 1 to n. Indeed,

$$\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 = \sum_{i,j} A_{i,j}\mathbf{u}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2 =$$
$$\|\mathbf{u}\|_2^2 - 2\langle A\mathbf{u}, \mathbf{v}\rangle + \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - 2\|\mathbf{v}\|_2^2 + \|\mathbf{v}\|_2^2\,,$$

where these equalities are due to the sum of each row and column in A equalling one, and because $\|\mathbf{v}\|_2^2 = \langle \mathbf{v}, \mathbf{v}\rangle = \langle A\mathbf{u}, \mathbf{v}\rangle = \sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j$.

Thus it suffices to show $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \frac{1}{d4n^3}$. This is a sum of non-negative terms so it suffices to show that for *some* i, j, $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \frac{1}{d4n^3}$. First, because we have all the self-loops, $A_{i,i} \geq 1/d$ for all i, and so we can assume $|\mathbf{u}_i - \mathbf{v}_i| < \frac{1}{2n^{1.5}}$ for every $i \in [n]$, as otherwise we'd be done.

Now sort the coordinates of $\mathbf{u}$ from the largest to the smallest, ensuring that $\mathbf{u}_1 \geq \mathbf{u}_2 \geq \cdots \mathbf{u}_n$. Since $\sum_i \mathbf{u}_i = 0$ it must hold that $\mathbf{u}_1 \geq 0 \geq \mathbf{u}_n$. In fact, since $\mathbf{u}$ is a unit vector, either $\mathbf{u}_1 \geq 1/\sqrt{n}$ or $\mathbf{u}_n \leq 1/\sqrt{n}$ and so $\mathbf{u}_1 - \mathbf{u}_n \geq 1/\sqrt{n}$. One of the $n-1$ differences between consecutive coordinates $\mathbf{u}_i - \mathbf{u}_{i+1}$ must be at least $1/n^{1.5}$ and so there must be an i_0 such that if we let $S = \{1, \ldots, i_0\}$ and $\overline{S} = [n] \setminus S_i$, then for every $i \in S$ and $j \in \overline{S}$, $\mathbf{u}_i - \mathbf{u}_j \geq 1/n^{1.5}$. Since G is connected there exists an edge (i, j) between S and $\overline{S}$. Since $|\mathbf{v}_j - \mathbf{u}_j| \leq \frac{1}{2n^{1.5}}$, for this choice of i, j, $|\mathbf{u}_i - \mathbf{v}_j| \geq |\mathbf{u}_i - \mathbf{u}_j| - \frac{1}{2n^{1.5}} \geq \frac{1}{2n^{1.5}}$. Thus $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \frac{1}{d}\frac{1}{4n^3}$. ■

Remark 7.29
The proof can be strengthened to show a similar result for every connected non-bipartite graph (not just those with self-loops at every vertex). Note that this condition is essential: if A is the adjacency matrix of a bipartite graph then one can find a vector $\mathbf{v}$ such that $A\mathbf{v} = -\mathbf{v}$.

7.A.2 Analysis of the randomized algorithm for undirected connectivity.

Together, Lemmas 7.27 and 7.28 imply that, at least for regular graphs, if s is connected to t then a sufficiently long random walk from s will hit t in polynomial time with high probability.

Corollary 7.30
Let G be a d-regular n-vertex graph with all vertices having a self-loop. Let s be a vertex in G. Let $T > 10dn^3 \log n$ and let X_T denote the distribution of the vertex of the T^{th} step in a random walk from s. Then, for every j connected to s, $\Pr[X_T = j] > \frac{1}{2n}$.

Proof: By these Lemmas, if we consider the restriction of an n-vertex graph G to the connected component of s, then for every probability vector $\mathbf{p}$ over this component and $T \geq 10dn^3 \log n$, $\|A^T\mathbf{p} - \mathbf{1}\|_2 < \frac{1}{2n^{1.5}}$ (where $\mathbf{1}$ here is the uniform distribution over this component). Using the relations between the L_1 and L_2 norms (see Note 7.24), $|A^T\mathbf{p} - \mathbf{1}|_1 < \frac{1}{2n}$ and hence every element in the connected component appears in $A^T\mathbf{p}$ with at least $1/n - 1/(2n) \geq 1/(2n)$ probability. ■

Note that Corollary 7.30 implies that if we repeat the $10dn^3 \log n$ walk for $10n$ times (or equivalently, if we take a walk of length $100dn^4 \log n$) then we will hit t with probability at least 3/4.

7.B Expander graphs.

Expander graphs are extremely useful combinatorial objects, which we will encounter several times in the book. They can be defined in two equivalent ways. At a high level, these two equivalent definitions can be described as follows:

- *Combinatorial definition:* A constant-degree regular graph G is an *expander* if for every subset S of less than half of G's vertices, a constant fraction of the edges touching S are from S to its complement in G. This is the definition alluded to in Section 7.5 (see Figure 7.1).[4]

- *Algebraic expansion:* A constant-degree regular graph G is an *expander* if its parameter $\lambda(G)$ bounded away from 1 by some constant. That is, $\lambda(G) \leq 1 - \epsilon$ for some constant $\epsilon > 0$¿

What do we mean by a constant? By *constant* we refer to a number that is independent of the size of the graph. We will typically talk about graphs that are part of an infinite *family* of graphs, and so by constant we mean a value that is the same for all graphs in the family, regardless of their size.

Below we make the definitions more precise, and show their equivalence. We will then complete the analysis of the randomness efficient error reduction procedure described in Section 7.5.

7.B.1 The Algebraic Definition

.

The Algebraic definition of expanders is as follows:

Definition 7.31 ((n, d, λ)-graphs.)
If G is an n-vertex d-regular G with $\lambda(G) \leq \lambda$ for some number $\lambda < 1$ then we say that G is an (n, d, λ)-graph.
A family of graphs $\{G_n\}_{n\in\mathbb{N}}$ is an *expander graph family* if there are some constants $d \in \mathbb{N}$ and $\lambda < 1$ such that for every n, G_n is an (n, d, λ)-graph.

Explicit constructions. We say that an expander family $\{G_n\}_{n\in\mathbb{N}}$ is *explicit* if there is a polynomial-time algorithm that on input 1^n with $n \in I$ outputs the adjacency matrix of G_n. We say that the family is *strongly explicit* if there is a polynomial-time algorithm that for every $n \in I$ on inputs $\langle n, v, i\rangle$ where $1 \leq v \leq n'$ and $1 \leq i \leq d$ outputs the i^{th} neighbor of v. (Note that the algorithm runs in time polynomial in the its input length which is polylogarithmic in n.)

As we will see below it is not hard to show that expander families exist using the probabilistic method. But this does not yield *explicit* (or very explicit) constructions of such graphs (which, as we saw in Section 7.4.1 are often needed for applications). In fact, there are also several explicit and

[4]The careful reader might note that there we said that a graph is an expander if a constant fraction of S's neighboring *vertices* are outside S. However, for constant-degree graphs these two notions are equivalent.

NOTE 7.33 (EXPLICIT CONSTRUCTION OF PSEUDORANDOM OBJECTS)
Expanders are one instance of a recurring theme in complexity theory (and other areas of math and computer science): it is often the case that a random object can be easily proven to satisfy some nice property, but the applications require an *explicit* object satisfying this property. In our case, a random d-regular graph is an expander, but to use it for, say, reducing the error of probabilistic algorithms, we need an *explicit* construction of an expander family, with an efficient deterministic algorithm to compute the neighborhood relations. Such explicit constructions can be sometimes hard to come by, but are often surprisingly useful. For example, in our case the explicit construction of expander graphs turns out to yield a deterministic logspace algorithm for undirected connectivity.
We will see another instance of this theme in Chapter 17, which discusses *error correcting codes*.

strongly explicit constructions of expander graphs known. The smallest λ can be for a d-regular n-vertex graph is $\Omega(\frac{1}{\sqrt{d}})$ and there are constructions meeting this bound (specifically the bound is $(1-o(1))\frac{2\sqrt{d-1}}{d}$ where by $o(1)$ we mean a function that tends to 0 as the number of vertices grows; graphs meeting this bound are called *Ramanujan graphs*). However, for most applications in Computer Science, any family with constant d and $\lambda < 1$ will suffice (see also Remark 7.32 below). Some of these constructions are very simple and efficient, but their analysis is highly non-trivial and uses relatively deep mathematics.[5] In Chapter 16 we will see a strongly explicit construction of expanders with elementary analysis. This construction also introduces a tool that is useful to derandomize the random-walk algorithm for UPATH.

REMARK 7.32
One reason that the particular constants of an expander family are not extremely crucial is that we can improve the constant λ (make it arbitrarily smaller) at the expense of increasing the degree: this follows from the fact, observed above in the proof of Lemma 7.27, that $\lambda(G^T) = \lambda(G)^T$, where G^T denotes the graph obtained by taking the adjacency matrix to the T^{th} power, or equivalently, having an edge for every length-T path in G. Thus, we can transform an (n, d, λ) graph into an (n, d^T, λ^T)-graph for every $T \geq 1$. In Chapter 16 we will see a different transformation called the *replacement product* to decrease the degree at the expense of increasing λ somewhat (and also increasing the number of vertices).

[5] An example for such an expander is the following 3-regular graph: the vertices are the numbers 1 to $p-1$ for some prime p, and each number x is connected to $x+1$,$x-1$ and x^{-1} (mod p).

7.B.2 Combinatorial expansion and existence of expanders.

We describe now a combinatorial criteria that is roughly equivalent to Definition 7.31. One advantage of this criteria is that it makes it easy to prove that a non-explicit expander family exists using the probabilistic method. It is also quite useful in several applications.

Definition 7.34 ([)
Combinatorial (edge) expansion] An n-vertex d-regular graph $G = (V, E)$ is called an (n, d, ρ)-*combinatorial expander* if for every subset $S \subseteq V$ with $|S| \leq n/2$, $|E(S, \overline{S})| \geq \rho d|S|$, where for subsets S, T of V, $E(S, T)$ denotes the set of edges (s, t) with $s \in S$ and $t \in T$.

Note that in this case the bigger ρ is the better the expander. We'll loosely use the term expander for any (n, d, ρ)-combinatorial expander with c a positive constant. Using the probabilistic method, one can prove the following theorem: (Exercise 12 asks you to prove a slightly weaker version)

Theorem 7.35 (Existence of expanders)
Let $\epsilon > 0$ be some constant. Then there exists $d = d(\epsilon)$ and $N \in \mathbb{N}$ such that for every $n > N$ there exists an $(n, d, 1 - \epsilon)$-combinatorial expander.

The following theorem related combinatorial expansion with our previous Definition 7.31

Theorem 7.36 (Combinatorial and algebraic expansion)

1. *If G is an (n, d, λ)-graph then it is an $(n, d, (1 - \lambda)/2)$-combinatorial expander.*
2. *If G is an (n, d, ρ)-combinatorial expander then it is an $(n, d, 1 - \frac{\rho^2}{2})$-graph.*

The first part of Theorem 7.36 follows by plugging $T = \overline{S}$ into the following lemma:

Lemma 7.37 (Expander Mixing Lemma)
Let $G = (V, E)$ be an (n, d, λ)-graph. Let $S, T \subseteq V$, then

$$\left| |E(S, T)| - \frac{d}{n}|S||T| \right| \leq \lambda d\sqrt{|S||T|}$$

Proof: Let $\mathbf{s}$ denote the vector such that $\mathbf{s}_i$ is equal to 1 if $i \in S$ and equal to 0 otherwise, and let $\mathbf{t}$ denote the corresponding vector for the set S. Thinking of $\mathbf{s}$ as a row vector and of $\mathbf{t}$ as a column vector, the Lemma's statement is equivalent to

$$\left| \mathbf{s}A\mathbf{t} - \tfrac{|S||T|}{n} \right| \leq \lambda\sqrt{|S||T|}, \tag{2}$$

where A is G's normalized adjacency matrix. Yet by Lemma 7.40, we can write A as $(1 - \lambda)J + \lambda C$, where J is the matrix with all entries equal to $1/n$ and C has norm at most one. Hence,

$$\mathbf{s}A\mathbf{t} = (1 - \lambda)\mathbf{s}J\mathbf{t} + \lambda\mathbf{s}C\mathbf{t} \leq \tfrac{|S||T|}{n} + \lambda\sqrt{|S||T|},$$

where the last inequality follows from $\mathbf{s}J\mathbf{t} = |S||T|/n$ and $\mathbf{s}C\mathbf{t} = \langle \mathbf{s}, C\mathbf{t}\rangle \le \|\mathbf{s}\|_2 \|\mathbf{t}\|_2 = \sqrt{|S||T|}$. ■

PROOF OF SECOND PART OF THEOREM 7.36.: We prove a slightly relaxed version, replacing the constant 2 with 8. Let $G = (V, E)$ be an n-vertex d-regular graph such that for every subset $S \subseteq V$ with $|S| \le n/2$, there are $\rho|S|$ edges between S and $\overline{S} = V \setminus S$, and let A be G's normalized adjacency matrix.

Let $\lambda = \lambda(G)$. We need to prove that $\lambda \le 1 - \rho^2/8$. Using the fact that λ is the second eigenvalue of A, there exists a vector $\mathbf{u} \perp \mathbf{1}$ such that $A\mathbf{u} = \lambda\mathbf{u}$. Write $\mathbf{u} = \mathbf{v} + \mathbf{w}$ where $\mathbf{v}$ is equal to $\mathbf{u}$ on the coordinates on which $\mathbf{u}$ is positive and equal to 0 otherwise, and $\mathbf{w}$ is equal to $\mathbf{u}$ on the coordinates on which $\mathbf{u}$ is negative, and equal to 0 otherwise. Note that, since $\mathbf{u} \perp \mathbf{1}$, both $\mathbf{v}$ and $\mathbf{w}$ are nonzero. We can assume that $\mathbf{u}$ is nonzero on at most $n/2$ of its coordinates (as otherwise we can take $-\mathbf{u}$ instead of $\mathbf{u}$).

Since $A\mathbf{u} = \lambda\mathbf{u}$ and $\langle \mathbf{v}, \mathbf{w}\rangle = 0$,

$$\langle A\mathbf{v}, \mathbf{v}\rangle + \langle A\mathbf{w}, \mathbf{v}\rangle = \langle A(\mathbf{v}+\mathbf{w}), \mathbf{v}\rangle = \langle A\mathbf{u}, \mathbf{v}\rangle = \langle \lambda(\mathbf{v}+\mathbf{w}), \mathbf{v}\rangle = \lambda\|\mathbf{v}\|_2^2 .$$

Since $\langle A\mathbf{w}, \mathbf{v}\rangle$ is negative, we get that $\langle A\mathbf{v}, \mathbf{v}\rangle / \|\mathbf{v}\|_2^2 \ge \lambda$ or

$$1 - \lambda \ge 1 - \frac{\langle A\mathbf{v}, \mathbf{v}\rangle}{\|\mathbf{v}\|_2^2} = \frac{\|\mathbf{v}\|_2^2 - \langle A\mathbf{v}, \mathbf{v}\rangle}{\|\mathbf{v}\|_2^2} = \frac{\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2}{2\|\mathbf{v}\|_2^2},$$

where the last equality is due to $\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2 = \sum_{i,j} A_{i,j}\mathbf{v}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{v}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2 = 2\|\mathbf{v}\|_2^2 - 2\langle A\mathbf{v}, \mathbf{v}\rangle$. (We use here the fact that each row and column of A sums to one.) Multiply both numerator and denominator by $\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 + \mathbf{v}_j^2)$. By the Cauchy-Schwartz inequality,[6] we can bound the new numerator as follows:

$$\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2\right)\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i + \mathbf{v}_j)^2\right) \le \left(\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)(\mathbf{v}_i + \mathbf{v}_j)\right)^2 .$$

Hence, using $(a-b)(a+b) = a^2 - b^2$,

$$1 - \lambda \ge \frac{\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)\right)^2}{2\|\mathbf{v}\|_2^2 \sum_{i,j} A_{i,j}(\mathbf{v}_i + \mathbf{v}_j)^2} = \frac{\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)\right)^2}{2\|\mathbf{v}\|_2^2 \left(\sum_{i,j} A_{i,j}\mathbf{v}_i^2 + 2\sum_{i,j} A_{i,j}\mathbf{v}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2\right)} =$$
$$\frac{\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)\right)^2}{2\|\mathbf{v}\|_2^2 \left(2\|\mathbf{v}\|_2^2 + 2\langle A\mathbf{v}, \mathbf{v}\rangle\right)} \ge \frac{\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)\right)^2}{8\|\mathbf{v}\|_2^4},$$

where the last inequality is due to A having matrix norm at most 1, implying $\langle A\mathbf{v}, \mathbf{v}\rangle \le \|\mathbf{v}\|_2^2$. We conclude the proof by showing that

$$\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2) \ge \rho\|\mathbf{v}\|_2^2 , \tag{3}$$

[6]The Cauchy-Schwartz inequality is typically stated as saying that for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\sum_i \mathbf{x}_i\mathbf{y}_i \le \sqrt{(\sum_i \mathbf{x}_i^2)(\sum_i \mathbf{y}_i^2)}$. However, it is easily generalized to show that for every non-negative $\mu_1, \ldots, \mu_n$, $\sum_i \mu_i\mathbf{x}_i\mathbf{y}_i \le \sqrt{(\sum_i \mu_i\mathbf{x}_i^2)(\sum_i \mu_i\mathbf{y}_i^2)}$ (this can be proven from the standard Cauchy-Schwartz by multiplying each coordinate of $\mathbf{x}$ and $\mathbf{y}$ by $\sqrt{\mu_i}$. It is this variant that we use here with the $A_{i,j}$'s playing the role of $\mu_1, \ldots, \mu_n$.

which indeed implies that $1-\lambda \geq \frac{\rho^2\|\mathbf{v}\|_2^4}{8\|\mathbf{v}\|_2^4} = \frac{\rho^2}{8}$.

To prove (3) sort the coordinates of $\mathbf{v}$ so that $\mathbf{v}_1 \geq \mathbf{v}_2 \geq \cdots \geq \mathbf{v}_n$ (with $\mathbf{v}_i = 0$ for $i > n/2$). Then

$$\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2) \geq \sum_{i=1}^{n/2}\sum_{j=i+1}^{n} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_{i+1}^2) = \sum_{i=1}^{n/2} c_i(\mathbf{v}_i^2 - \mathbf{v}_{i+1}^2)\,,$$

where c_i denotes $\sum_{j>i} A_{i,j}$. But c_i is equal to the number of edges in G from the set $\{k : k \leq i\}$ to its complement, divided by d. Hence, by the expansion of G, $c_i \geq \rho i$, implying (using the fact that $\mathbf{v}_i = 0$ for $i \geq n/2$):

$$\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2) \geq \sum_{i=1}^{n/2} \rho i(\mathbf{v}_i^2 - \mathbf{v}_{i+1}^2) = \sum_{i=1}^{n/2} (\rho i \mathbf{v}_i^2 - \rho\cdot(i-1)\mathbf{v}_i^2) = \rho\|\mathbf{v}\|_2^2\,,$$

establishing (3). ■

7.B.3 Error reduction using expanders.

We now complete the analysis of the randomness efficient error reduction procedure described in Section 7.5. Recall, that this procedure was the following: let $N = 2^m$ where m is the number of coins the randomized algorithm uses. We use $m + O(k)$ random coins to select a k-vertex random walk in an expander graph G_N, and then output 1 if and only if the algorithm outputs 1 when given one of the vertices in the walk as random coins. To show this procedure works we need to show that if the probabilistic algorithm outputs 1 for at least half of the coins, then the probability that all the vertices of the walk correspond to coins on which the algorithm outputs 0 is exponentially small in k. This will be a direct consequence of the following theorem: (think of the set B below as the set of vertices corresponding to coins on which the algorithm outputs 0)

Theorem 7.38 (Expander walks)
Let G be an (N,d,λ) graph, and let $B \subseteq [N]$ be a set with $|B| \leq \beta N$. Let $X_1,\ldots,X_k$ be random variables denoting a $k-1$-step random walk from X_1, where X_1 is chosen uniformly in $[N]$. Then,

$$\underset{(*)}{\Pr[\forall_{1\leq i\leq k} X_i \in B]} \leq ((1-\lambda)\sqrt{\beta}+\lambda)^{k-1}$$

Note that if λ and β are both constants smaller than 1 then so is the expression $(1-\lambda)\sqrt{\beta}+\lambda$.
Proof: For $1 \leq i \leq k$, let B_i be the event that $X_i \in B$. Note that the probability $(*)$ we're trying to bound is $\Pr[B_1]\Pr[B_2|B_1]\cdots\Pr[B_k|B_1,\ldots,B_{k-1}]$. Let $\mathbf{p}^i \in \mathbb{R}^N$ be the vector representing the distribution of X_i, conditioned on the events $B_1,\ldots,B_i$. Denote by $\hat{B}$ the following linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$: for every $\mathbf{u} \in \mathbb{R}^N$, and $j \in [N]$, $(\hat{B}\mathbf{u})_j = \mathbf{u}_j$ if $j \in B$ and $(\hat{B}\mathbf{u})_j = 0$ otherwise. It's not hard to verify that $\mathbf{p}^1 = \frac{1}{\Pr[B_1]}\hat{B}\mathbf{1}$ (recall that $\mathbf{1} = (1/N,\ldots,1/N)$ is the

vector representing the uniform distribution over $[N]$). Similarly, $\mathbf{p}^2 = \frac{1}{\Pr[B_2|B_1]}\frac{1}{\Pr[B_1]}\hat{B}A\hat{B}\mathbf{1}$ where $A = A(G)$ is the adjacency matrix of G. Since every probability vector $\mathbf{p}$ satisfies $|\mathbf{p}|_1 = 1$,

$$(*) = |(\hat{B}A)^{k-1}\hat{B}\mathbf{1}|_1$$

We bound this norm by showing that

$$\|(\hat{B}A)^{k-1}\hat{B}\mathbf{1}\|_2 \le \tfrac{((1-\lambda)\sqrt{\beta}+\lambda)^{k-1}}{\sqrt{N}} \tag{4}$$

which suffices since for every $\mathbf{v} \in \mathbb{R}^N$, $|\mathbf{v}|_1 \le \sqrt{N}\|\mathbf{v}\|_2$ (see Note 7.24).

To prove (4), we use the following definition and lemma:

Definition 7.39 (Matrix Norm)
If A is an m by n matrix, then $\|A\|$ is the maximum number α such that $\|A\mathbf{v}\|_2 \le \alpha\|\mathbf{v}\|_2$ for every $\mathbf{v} \in \mathbb{R}^n$.

Note that if A is a normalized adjacency matrix then $\|A\| = 1$ (as $A\mathbf{1} = \mathbf{1}$ and $\|A\mathbf{v}\|_2 \le \|\mathbf{v}\|_2$ for every $\mathbf{v}$). Also note that the matrix norm satisfies that for every two n by n matrices A, B, $\|A + B\| \le \|A\| + \|B\|$ and $\|AB\| \le \|A\|\|B\|$.

Lemma 7.40
Let A be a normalized adjacency matrix of an (n, d, λ)-graph G. Let J be the adjacency matrix of the n-clique with self loops (i.e., $J_{i,j} = 1/n$ for every i, j). Then

$$A = (1-\lambda)J + \lambda C \tag{5}$$

where $\|C\| \le 1$.

Note that for every probability vector $\mathbf{p}$, $J\mathbf{p}$ is the uniform distribution, and so this lemma tells us that in some sense, we can think of a step on a (n, d, λ)-graph as going to the uniform distribution with probability $1-\lambda$, and to a different distribution with probability λ. This is of course not completely accurate, as a step on a d-regular graph will only go the one of the d neighbors of the current vertex, but we'll see that for the purposes of our analysis, the condition (5) will be just as good.[7]

Proof of Lemma 7.40: Indeed, simply define $C = \frac{1}{\lambda}(A - (1-\lambda)J)$. We need to prove $\|C\mathbf{v}\|_2 \le \|\mathbf{v}\|_2$ for very $\mathbf{v}$. Decompose $\mathbf{v}$ as $\mathbf{v} = \mathbf{u} + \mathbf{w}$ where u is $\alpha\mathbf{1}$ for some α and $\mathbf{w} \perp \mathbf{1}$, and $\|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2$. Since $A\mathbf{1} = \mathbf{1}$ and $J\mathbf{1} = \mathbf{1}$ we get that $C\mathbf{u} = \frac{1}{\lambda}(\mathbf{u} - (1-\lambda)\mathbf{u}) = \mathbf{u}$. Now, let $\mathbf{w}' = A\mathbf{w}$. Then $\|\mathbf{w}'\|_2 \le \lambda\|\mathbf{w}\|_2$ and, as we saw in the proof of Lemma 7.27, $\mathbf{w}' \perp \mathbf{1}$. Furthermore, since the sum of the coordinates of $\mathbf{w}$ is zero, $J\mathbf{w} = \mathbf{0}$. We get that $C\mathbf{w} = \frac{1}{\lambda}\mathbf{w}'$. Since $\mathbf{w}' \perp \mathbf{u}$, $\|C\mathbf{w}\|_2^2 = \|\mathbf{u} + \frac{1}{\lambda}\mathbf{w}'\|_2^2 = \|\mathbf{u}\|_2^2 + \|\frac{1}{\lambda}\mathbf{w}'\|_2^2 \le \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2 = \|\mathbf{w}\|_2^2$. ■

Returning to the proof of Theorem 7.38, we can write $\hat{B}A = \hat{B}\big((1-\lambda)J + \lambda C\big)$, and hence $\|\hat{B}A\| \le (1-\lambda)\|\hat{B}J\| + \lambda\|\hat{B}C\|$. Since J's output is always a vector of the form $\alpha\mathbf{1}$, $\|\hat{B}J\| \le \sqrt{\beta}$. Also, because $\hat{B}$ is an operation that merely zeros out some parts of its input, $\|\hat{B}\| \le 1$ implying

[7]Algebraically, the reason (5) is not equivalent to going to the uniform distribution in each step with probability $1-\lambda$ is that C is not necessarily a stochastic matrix, and may have negative entries.

$\|\hat{B}C\| \leq 1$. Thus, $\|\hat{B}A\| \leq (1-\lambda)\sqrt{\beta}+\lambda$. Since $B\mathbf{1}$ has the value $1/N$ in $|B|$ places, $\|B\mathbf{1}\|_2 = \frac{\sqrt{\beta}}{\sqrt{N}}$, and hence $\|(\hat{B}A)^{k-1}\hat{B}\mathbf{1}\|_2 \leq ((1-\lambda)\sqrt{\beta}+\lambda)^{k-1}\frac{\sqrt{\beta}}{\sqrt{N}}$, establishing (4). ■

One can obtain a similar error reduction procedure for *two-sided error* algorithms by running the algorithm using the k sets of coins obtained from a $k-1$ step random walk and deciding on the output according to the *majority* of the values obtained. The analysis of this procedure is based on the following theorem, whose proof we omit:

Theorem 7.41 (Expander Chernoff Bound [**?**])
Let G be an (N,d,λ)-graph and $B \subseteq [N]$ with $|B| = \beta N$. Let $X_1,\ldots,X_k$ be random variables denoting a $k-1$-step random walk in G (where X_1 is chosen uniformly). For every $i \in [k]$, define B_i to be 1 if $X_i \in B$ and 0 otherwise. Then, for every $\delta > 0$,

$$\Pr\left[\left|\frac{\sum_{i=1}^k B_i}{k} - \beta\right| > \delta\right] < 2e^{(1-\lambda)\delta^2 k/60}$$