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 17 and 18), 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 the appendix 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 **BPP** $\subseteq$ **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, 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.

Formally, primality testing consists of checking membership in the language $\mathsf{PRIMES} = \{ \llcorner N \lrcorner : N \text{ is a prime number}\}$. 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 21 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

[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.

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 the appendix (see Lemma A.23):

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 $C \in$ ZEROP then we always accept. By the lemma, if $C \notin$ 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
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:

$$\begin{aligned} \mathbf{ZPP} &= \mathbf{RP} \cap \mathbf{coRP} \\ \mathbf{RP} &\subseteq \mathbf{BPP} \\ \mathbf{coRP} &\subseteq \mathbf{BPP} \end{aligned}$$

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.16 in the appendix) 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,

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

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 Chapter 16).

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.8), 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 $1/100$.

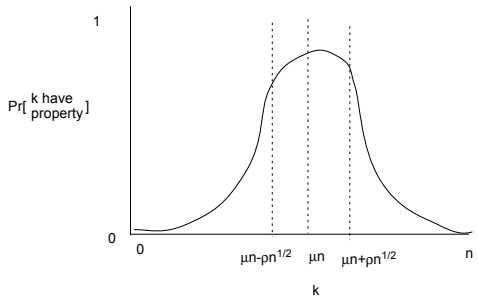
7.4.3 Allowing more general random choices than a fair random coin.

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 $b_i > p_i$

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 7.16, showing that **BPP** $\subseteq$ **P**/poly.

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 17, we do know how to simulate probabilistic algorithms designed for perfect independent 1/2-coins even using such a weak random source.

7.5 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** $= \Sigma_2^p$.

THEOREM 7.16 (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 $x \in \{0,1\}^n$. ■

7.6 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 7.17 (SIPSER-GÁCS)
BPP $\subseteq \mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$

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

Suppose $L \in$ **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.

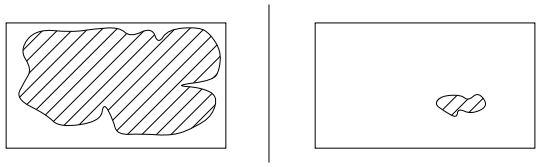


Figure 7.1: There are only two possible sizes for the set of r's such that $M(x,r)$ =Accept: either this set is almost all of $\{0,1\}^m$ or a tiny fraction of $\{0,1\}^m$. In the former case, a few random "shifts" of this set are quite likely to cover all of $\{0,1\}^m$. In the latter case the set's 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 $L \in \mathbf{\Sigma_2}$. ■

7.7 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 17 and 18). 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 4. 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).

7.8 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 8 and 9).

DEFINITION 7.18
Language A reduces to language B under a randomized polynomial time reduction, denoted $A \leq_r B$, if there exists a deterministic, polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ and a polynomial $p()$ such that

$$\forall x \in A \qquad \mathbf{Pr}_{y \in \{0,1\}^{p(|x|)}}[f(x,y) \in B] \geq 2/3$$
$$\forall x \notin A \qquad \mathbf{Pr}_{y \in \{0,1\}^{p(|x|)}}[f(x,y) \in B] \leq 1/3$$

We note that if $A \leq_r B$ 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 $\{L : L \leq_p 3\mathsf{SAT}\}$. The following definition is analogous.

DEFINITION 7.19 ($\mathbf{BP} \cdot \mathbf{NP}$)
$\mathbf{BP} \cdot \mathbf{NP} = \{L : L \leq_r 3\mathsf{SAT}\}$.

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

One interesting application of randomized reductions will be shown in Chapter 8, 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.

7.9 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 worktape has $O(\log n)$ size and the associated complexity classes **RL** and **BPL** are defined analogously to **RP** and **BPP**. That is, $L \in$ **BPL** if there's a $O(\log n)$-space PTM M such that for every $x \in \{0,1\}^*$, $\Pr[M(x) = L(x)] \geq 2/3$, whereas $L \in$ **RL** if there's a $O(\log n)$-space PTM M such that **(1)** for every $x \in L$, $\Pr[M(x) = L(x)] \geq 2/3$ and **(2)** for every $x \notin L$, $\Pr[M(x) = L(x)] = 0$. 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 **RL** $\subseteq$ **NL**, and thus **RL** $\subseteq$ **P**. The exercises ask you to show that **BPL** $\subseteq$ **P** as well.

One famous **RL**-algorithm is the algorithm to solve UPATH: the restriction of the **NL**-complete PATH problem (see Chapter 3) 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. The algorithm 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 $\frac{4}{27}n^3$ (see Exercise 9 for a somewhat weaker bound) and hence our algorithm will accept with probability at least $\frac{3}{4}$. In Chapter 16 we analyze a variant of this algorithm and also show a recent *deterministic* logspace algorithm for the same problem.

It is known that **BPL** (and hence also **RL**) is contained in **SPACE**$(\log^{3/2} n)$. In Chapter 17 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 $\mathbf{P}/\mathrm{poly}$ 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 [**?**, **?**].

$\mathbf{BPP} \subseteq \mathbf{P}/\text{poly}$ 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 $\mathbf{BPP}/1$ (i.e., **BPP** with a single bit of nonuniform advice) [**?**, **?**, **?**].

Readers interested in randomized algorithms are referred to the excellent book by Motwani and Raghavan [**?**] from the mid 1990s.

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 $\Delta(\mathbf{p}^k) \leq (1 - n^{-10n})\Delta(\mathbf{p})$. 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 $E_u \leq 10n^2$.

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 $E_{u,v} \leq 100kn^2$. 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 $\sum_{m \in \mathbb{N}} \Pr[X \geq m]$ 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 $1/2$.