Texts in Computational Complexity: Randomized Complexity Classes

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

January 5, 2006

So far (in the course), our approach to computing devices was somewhat conservative: we thought of them as executing a deterministic rule. A more liberal and quite realistic approach, which is pursued in this chapter, considers computing devices that use a probabilistic rule. This relaxation has an immediate impact on the notion of efficient computation, which is consequently associated with *probabilistic* polynomial-time computations rather than with deterministic (polynomial-time) ones. We stress that the association of efficient computation with probabilistic polynomial-time computation makes sense provided that the failure probability of the latter is negligible (which means that it may be safely ignored).

Focusing on probabilistic polynomial-time algorithms, we consider various types of failure of such algorithms yielding complexity classes such as $\mathcal{BPP}$, $\mathcal{RP}$, and $\mathcal{ZPP}$. The results presented include $\mathcal{BPP} \subset \mathcal{P}/\text{poly}$ and $\mathcal{BPP} \subseteq \Sigma_2$.

1 The general setting

Considering algorithms that utilize random choices, we extend our notion of *efficient algorithms* from *deterministic* polynomial-time algorithms to *probabilistic* polynomial-time algorithms.

Rigorous models of probabilistic (or randomized) algorithms are defined by natural extensions of the basic machine model. We will exemplify this approach by describing the model of probabilistic Turing machines, but we stress that (again) the specific choice of the model is immaterial (as long as it is "reasonable"). A probabilistic Turing machine is defined exactly as a non-deterministic machine, *but the definition of its computation is fundamentally different.* Specifically, whereas the definition of a non-deterministic machine refers to the question of whether or not there exists a computation of the machine that (started on a specific input) reaches a certain configuration, in case of probabilistic Turing machines we refer to *the probability that this event occurs, when at each step a choice is selected uniformly among the relevant possible choices available at this step.* That is, if the transition function of the machine maps the current state-symbol pair to several possible triples, then the probabilistic machine randomly selects one of these triples (with equal probability) and the next configuration is determined accordingly. These random choices may be viewed as the internal coin tosses of the machine. (Indeed, as in the case of non-deterministic machines, we may assume without loss of generality that the transition function of the machine maps each state-symbol pair to exactly two possible triples.)

We stress the fundamental difference between the fictitious model of a non-deterministic machine and the realistic model of a probabilistic machine. In the case of a non-deterministic machine we consider the *existence* of an adequate sequence of choices (leading to a desired outcome), and ignore the question of how these choices are actually made. In fact, the selection of such a sequence of choices is merely a mental experiment. In contrast, in the case of a probabilistic machine, at each step a choice is made uniformly (among a set of predetermined possibilities), and we consider the *probability* of reaching a desired outcome.

In view of the foregoing, we consider the output distribution of such a probabilistic machine on fixed inputs; that is, for a probabilistic machine M and string $x \in \{0,1\}^*$, we denote by $M(x)$ the output distribution of M when invoked on input x, where the probability is taken uniformly over the machine's internal coin tosses. Needless to say, we will consider the probability that $M(x)$ is a "correct" answer; that is, in the case of a search problem (resp., decision problem) we will be interested in the probability that $M(x)$ is a valid solution for the instance x (resp., represents the correct decision regarding x).

The foregoing description views the internal coin tosses of the machine as taking place on-the-fly; that is, these coin tosses are performed *on-line* by the machine itself. An alternative model is one in which the sequence of coin tosses is provided by an external device, on a special "random input" tape. In such a case, we view these coin tosses as performed *off-line*. Specifically, we denote by $M'(x,r)$ the (uniquely defined) output of the residual deterministic machine M', when given the (primary) input x and random input r. Indeed, M' is a deterministic machine that takes two inputs (the first representing the actual input and the second representing the "random input"), but we consider the random variable $M(x) \stackrel{\text{def}}{=} M'(x, U_{\ell(|x|)})$, where $\ell(|x|)$ denotes the number of coin tosses "expected" by $M'(x,\cdot)$.

These two perspectives on probabilistic algorithms are clearly related: The computation of an on-line machine M is captured by the residual machine M', which may be viewed as receiving a sequence of coin tosses (obtained off-line), where the number of these coin tosses is determined by the time-complexity of M. (Indeed, there is no harm in supplying more coin tosses than are actually used by M.) For sake of clarity and future reference, we state the following definition.

Definition 1 (on-line and off-line formulations of probabilistic polynomial-time):

- *We say that M is a* on-line probabilistic polynomial-time machine *if there exists a polynomial p such that when invoked on any input $x \in \{0,1\}^*$, machine M always halts within at most $p(|x|)$ steps* (regardless of the outcome of its internal coin tosses). *In such a case $M(x)$ is a random variable.*

- *We say that M' is a* off-line probabilistic polynomial-time machine *if there exists a polynomial p such that, for every $x \in \{0,1\}^*$ and $r \in \{0,1\}^{p(|x|)}$, when invoked on the* primary input *x and the* random-input *sequence r, machine M' halts within at most $p(|x|)$ steps. In such a case, we will consider the random variable $M'(x, U_{p(|x|)})$.*

Clearly, the on-line and off-line formulations are equivalent (i.e., given a on-line probabilistic polynomial-time machine we can derive a functionally equivalent off-line (probabilistic polynomial-time) machine, and vice versa). Thus, in the sequel, we will freely use whichever is more convenient.

Failure probability. A major aspect of randomized algorithms (probabilistic machines) is that they may fail (see Exercise 12). That is, with some specified ("failure") probability, these algorithms may fail to produce the desired output. We discuss two aspects of this failure: its *type* and its *magnitude*.

1. The type of failure is a qualitative notion. One aspect of this type is whether, in case of failure, the algorithm produces a wrong answer or merely an indication that it failed to find a correct answer. Another aspect is whether failure may occur on all instances or merely on certain types of instances. Let us clarify these aspects by considering three natural types of failure, giving rise to three different types of algorithms.

   (a) The most liberal notion of failure is the one of two-sided error. This term originates from the setting of decision problems, where it means that (in case of failure) the algorithm may err in both directions (i.e., it may rule that a yes-instance is a no-instance, and vice versa). In the case of search problems two-sided error means that, when failing, the algorithm may output a wrong answer on any input. Furthermore, the algorithm may falsely rule that the input has no solution and it may also output a wrong solution (both in case the input has a solution and in case it has no solution).

   (b) An intermediate notion of failure is the one of one-sided error. Again, the term originates from the setting of decision problems, where it means that the algorithm may err only in one direction (i.e., either on yes-instances or on no-instances). Indeed, there are two natural cases depending on whether the algorithm errs on yes-instances (but not on no-instances), or the other way around. Analogous cases occur also in the setting of search problems. In one case the algorithm never outputs a wrong solution but may falsely rule that the input has no solution. In the other case the indication that an input has no solution is never wrong, but the algorithm may output a wrong solution.

   (c) The most conservative notion of failure is the one of zero-sided error. In this case, the algorithm's failure amounts to indicating its failure to find an answer (by outputting a special `don't know` symbol). We stress that in this case the algorithm *never provides a wrong answer*.

   Indeed, the forgoing discussion ignores the probability of failure, which is the subject of the next item.

2. The magnitude of failure is a quantitative notion. It refer to the probability that the algorithm fails, where the type of failure is fixed (e.g., as in the forgoing discussion).

   When actually using a randomized algorithm we typically wish its failure probability to be negligible, which intuitively means that the failure event is so rare that it can be ignored in practice. Formally, we say that a quantity is negligible if, as a function of the relevant parameter (e.g., the input length), this quantity vanishes faster than the reciprocal of any positive polynomial.

   For ease of presentation, we sometimes consider alternative upper-bounds on the probability of failure. These bounds are selected in a way that allows (and in fact facilitates) "error reduction" (i.e., converting a probabilistic polynomial-time algorithm that satisfies such an upper-bound into one in which the failure probability is negligible). For example, in case of two-sided error we need to be able to distinguish the correct answer from wrong answers by sampling, and in the other types of failure "hitting" a correct answer suffices.

In the following three sections, we will discuss complexity classes corresponding to the aforementioned three types of failure. For sake of simplicity, the failure probability itself will be set to a constant that allows error reduction.

Randomized reductions. Before turning to the more detailed discussion, we note that randomized reductions play an important role in complexity theory. Such reductions can be defined analogously to the standard Cook-Reductions (resp., Karp-reduction), and again a discussion of the type and magnitude of the failure probability is in place. For clarity, we spell-out the two-sided error versions.

- In analogy to the definition of Cook reductions, we say that a problem Π is probabilistic polynomial-time reducible to a problem Π' if there exists a probabilistic polynomial-time oracle machine M such that, for every function f that solves Π' and for every x, with probability at least $1 - \mu(|x|)$, the output $M^f(x)$ is a correct solution to the instance x, where μ is a negligible function.
- In analogy to the definition of Karp reductions, we say that a decision problem S is reducible to a decision problem S' via a randomized Karp-reduction if there exists a probabilistic polynomial-time algorithm A such that, for every x, it holds that $\Pr[\chi_{S'}(A(x)) = \chi_S(x)] \geq 1 - \mu(|x|)$, where χ_S (resp., $\chi_{S'}$) is the characteristic function of S (resp., S') and μ is a negligible function.

These reductions preserve efficient solvability and are transitive: see Exercise 13.

2 Two-sided error: The complexity class BPP

In this section we consider the most liberal notion of probabilistic polynomial-time algorithms that is still meaningful. We allow the algorithm to err on each input, but require the error probability to be *negligible*. The latter requirement guarantees the usefulness of such algorithms, because in reality we may ignore the negligible error probability.

Before focusing on the decision problem setting, let us say a few words on the search problem setting. Following the previous paragraph, we say that a probabilistic (polynomial-time) algorithm A solves the search problem of the relation R if for every $x \in S_R$ (i.e., $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\} \neq \emptyset$) it holds that $\Pr[A(x) \in R(x)] > 1 - \mu(|x|)$ and for every $x \notin S_R$ it holds that $\Pr[A(x) = \perp] > 1 - \mu(|x|)$, where μ is a negligible function. Note that we did not require that, when invoked on input x that has a solution (i.e., $R(x) \neq \emptyset$), the algorithm always outputs the same solution. That is, a stronger requirement is that for every such x there exists $y \in R(x)$ such that $\Pr[A(x) = y] > 1 - \mu(|x|)$. The latter version as well as quantitative relaxations of it allow for error-reduction (see Exercise 14), discussed next.

Turning to decision problems, we consider probabilistic polynomial-time algorithms that err with negligible probability. That is, we say that a probabilistic (polynomial-time) algorithm A decides membership in S if for every x it holds that $\Pr[A(x) = \chi_S(x)] > 1 - \mu(|x|)$, where χ_S is the characteristic function of S (i.e., $\chi_S(x) = 1$ if $x \in S$ and $\chi_S(x) = 0$ otherwise) and μ is a negligible function. The class of decision problems that are solvable by probabilistic polynomial-time algorithms is denoted $\mathcal{BPP}$, standing for Bounded-error Probabilistic Polynomial-time. Actually, the standard definition refers to machines that err with probability at most 1/3.

Definition 2 (the class $\mathcal{BPP}$): *A decision problem S is in $\mathcal{BPP}$ if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that $\Pr[A(x) = 1] \geq 2/3$ and for every $x \notin S$ it holds that $\Pr[A(x) = 0] \geq 2/3$.*

The choice of the constant 2/3 is immaterial, and any other constant greater than 1/2 will do (and yields the very same class). Similarly, the complementary constant 1/3 can be replaced by various negligible functions (while preserving the class). Both facts are special cases of the robustness of the class, which is established using the process of error reduction.

Error reduction (or confidence amplification). For $\varepsilon : \mathbb{N} \to (0, 0.5)$, let $\mathcal{BPP}_\varepsilon$ denote the class of decision problems that can be solved in probabilistic polynomial-time with error probability upper-bounded by ε; that is, $S \in \mathcal{BPP}_\varepsilon$ if there exists a probabilistic polynomial-time algorithm A such that for every x it holds that $\Pr[A(x) \neq \chi_S(x)] \leq \varepsilon(|x|)$. By definition, $\mathcal{BPP} = \mathcal{BPP}_{1/3}$. However, a wide range of other classes also equal $\mathcal{BPP}$. In particular, we mention two extreme cases:

1. For every positive polynomial p and $\varepsilon(n) = (1/2) - (1/p(n))$, the class $\mathcal{BPP}_\varepsilon$ equals $\mathcal{BPP}$. That is, any error that is ("noticeably") bounded away from 1/2 (i.e., error $(1/2) - (1/\text{poly}(n))$) can be reduced to an error of 1/3.
2. For every positive polynomial p and $\varepsilon(n) = 2^{-p(n)}$, the class $\mathcal{BPP}_\varepsilon$ equals $\mathcal{BPP}$. That is, an error of 1/3 can be further reduced to an exponentially vanishing error.

Both facts are proved by invoking the weaker algorithm (i.e., the one having a larger error probability bound) for an adequate number of times, and ruling by majority. We stress that invoking a randomized machine several times means that the random choices made in the various invocations are independent of one another. The success probability of such a process is analyzed by applying an adequate Law of Large Numbers (see Exercise 15).

2.1 On the power of randomization

A natural question arises: *Did we gain anything in extending the definition of efficient computation to include also probabilistic polynomial-time ones?*

This phrasing seems too generic. We certainly gained the ability to toss coins (and generate various distributions). More concretely, randomized algorithms are essential in many settings (e.g., cryptography, probabilistic proof systems, and property testing) and seem essential in others (e.g., studies of the complexity of finding unique solutions and approximate counting). What we mean to ask here is *whether allowing randomization increases the power of polynomial-time algorithms also in the restricted context of solving decision and search problems?*

The question is whether $\mathcal{BPP}$ extends beyond $\mathcal{P}$ (where clearly $\mathcal{P} \subseteq \mathcal{BPP}$). It is commonly conjectured that the answer is negative. Specifically, under some reasonable assumptions, it holds that $\mathcal{BPP} = \mathcal{P}$. We note, however, that a polynomial slow-down occurs in the proof of the latter result; that is, randomized algorithms that run in time $t(\cdot)$ are emulated by deterministic algorithms that run in time $\text{poly}(t(\cdot))$. Furthermore, for some concrete problems (most notably primality testing (cf. Section 2.2)), the known probabilistic polynomial-time algorithm is significantly faster (and conceptually simpler) than the known deterministic polynomial-time algorithm. Thus, we believe that even in the context of decision problems, the notion of probabilistic polynomial-time algorithms is advantageous. We note that the fundamental nature of $\mathcal{BPP}$ will hold even in the (rather unlikely) case that it turns out that it offers no computational advantage (i.e., even if every problem that can be decided in probabilistic polynomial-time can be decided by a deterministic algorithm of essentially the same complexity).[1]

[1] Such a result would address a fundamental question regarding the power of randomness. By analogy, establishing that $\mathcal{IP} = \mathcal{PSPACE}$ does not diminish the importance of any of these classes.

BPP is in the Polynomial-Time Hierarchy: While it may be that $\mathcal{BPP} = \mathcal{P}$, it is not known whether or not $\mathcal{BPP}$ is contained in $\mathcal{NP}$. The source of trouble is the two-sided error probability of $\mathcal{BPP}$, which is incompatible with the absolute rejection of no-instances required in the definition of $\mathcal{NP}$ (see Exercise 22). In view of this ignorance, it is interesting to note that $\mathcal{BPP}$ resides in the second level of the Polynomial-Time Hierarchy (i.e., $\mathcal{BPP} \subseteq \Sigma_2$). This is a corollary of Theorem 7.

Trivial derandomization. A straightforward way of eliminating randomness from an algorithm is trying all possible outcomes of its internal coin tosses, collecting the relevant statistics and deciding accordingly. This yields $\mathcal{BPP} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP}$, which is considered the trivial derandomization of $\mathcal{BPP}$. In later stages of this course, we will consider various non-trivial derandomizations of $\mathcal{BPP}$, which are known under various intractability assumptions. The interested reader, who may be puzzled by the connection between derandomization and computational difficulty, is referred to these later lectures.

Non-uniform derandomization. In many settings (and specifically in the context of solving search and decision problems), the power of randomization is superseded by the power of non-uniform advice. Intuitively, the non-uniform advice may specify a sequence of coin tosses that is good for all (primary) inputs of a specific length. In the context of solving search and decision problems, such an advice must be good for *each* of these inputs[2], and thus its existence is guaranteed only if the error probability is low enough (so as to support a union bound). The latter condition can be guaranteed by error-reduction, and thus we get the following result.

Theorem 3 *$\mathcal{BPP}$ is* (strictly) *contained in* $\mathcal{P}$/poly.

Proof: Recall that $\mathcal{P}$/poly contains undecidable problems, which are certainly not in $\mathcal{BPP}$. Thus, we focus on showing that $\mathcal{BPP} \subseteq \mathcal{P}$/poly. By the discussion regarding error-reduction, for every $S \in \mathcal{BPP}$ there exists a (deterministic) polynomial-time algorithm A and a polynomial p such that for every x it holds that $\mathsf{Pr}[A(x, U_{p(|x|)}) \neq \chi_S(x)] < 2^{-|x|}$. Using a union bound, it follows that $\mathsf{Pr}_{r \in \{0,1\}^{p(n)}}[\exists x \in \{0,1\}^n \text{ s.t. } A(x,r) \neq \chi_S(x)] < 1$. Thus, for every $n \in \mathbb{N}$, there exists a string $r_n \in \{0,1\}^{p(n)}$ such that for every $x \in \{0,1\}^n$ it holds that $A(x, r_n) = \chi_S(x)$. Using such a sequence of r_n's as advice, we obtain the desired non-uniform machine (establishing $S \in \mathcal{P}$/poly). ■

2.2 A probabilistic polynomial-time primality test

Teaching note: Although primality has been recently shown to be in $\mathcal{P}$, we believe that the following example provides a nice illustration to the power of randomized algorithms.

We present a simple probabilistic polynomial-time algorithm for deciding whether or not a given number is a prime. The only Number Theoretic facts that we use are:

Fact 1: For every prime $p > 2$, each quadratic residue mod p has exactly two square roots mod p (and they sum-up to p).[3]

Fact 2: For every (odd and non-integer-power) composite number N, each quadratic residue mod N has at least four square roots mod N.

[2] In other contexts (e.g., one-way functions and pseudorandomness), it suffices to have an advice that is good on the average, where the average is taken over all relevant (primary) inputs.

[3] That is, for every $r \in \{1, ..., p-1\}$, the equation $x^2 \equiv s^2 \pmod{p}$ has two solutions modulo p (i.e., s and $p - s$).

Our algorithm uses as a black-box an algorithm, denoted `sqrt`, that given a prime p and a quadratic residue mod p, denoted s, returns the smallest among the two modular square roots of s. There is no guarantee as to what the output is in the case that the input is not of the aforementioned form (and in particular in the case that p is not a prime). Thus, we actually present a probabilistic polynomial-time reduction of testing primality to extracting square roots modulo a prime (which is a search problem with a promise).

Construction 4 (the reduction): *On input a natural number* $N > 2$ *do*

1. *If* N *is either even or an integer-power*[4] *then* `reject`.
2. *Uniformly select* $r \in \{1, ..., N-1\}$, *and set* $s \leftarrow r^2 \bmod N$.
3. *Let* $r' \leftarrow \texttt{sqrt}(s, N)$. *If* $r' \equiv \pm r \pmod N$ *then* `accept` *else* `reject`.

Indeed, in the case that N is composite, the reduction invokes `sqrt` on an illegitimate input (i.e., it makes a query that violates the promise of the problem at the target of the reduction). In such a case, there is not guarantee as to what `sqrt` answers, but actually a bluntly wrong answer only plays in our favor. In general, we will show that if N is composite, then the reduction rejects with probability at least 1/2, regardless of how `sqrt` answers. We mention that there exists a probabilistic polynomial-time algorithm for implementing `sqrt` (see Exercise 25).

Proposition 5 *Construction 4 constitutes a probabilistic polynomial-time reduction of testing primality to extracting square roots module a prime. Furthermore, if the input is a prime then the reduction always accepts, and otherwise it rejects with probability at least* 1/2.

We stress that Proposition 5 refers to the reduction itself; that is, `sqrt` is viewed as a ("perfect") oracle that, for every prime P and quadratic residue $s \pmod P$, returns $r < s/2$ such that $r^2 \equiv s \pmod P$. Combining Proposition 5 with a probabilistic polynomial-time algorithm that computes `sqrt` with negligible error probability, we obtain that testing primality is in $\mathcal{BPP}$.

Proof: By Fact 1, on input a prime number N, Construction 4 always accepts (because in this case, for every $r \in \{1, ..., N-1\}$, it holds that $\texttt{sqrt}(r^2 \bmod N, N) = \pm r$). On the other hand, suppose that N is an odd composite that is not an integer-power. Then, by Fact 2, each quadratic residue s has at least four square roots, and each of these square roots is equally likely to be chosen at Step 2 (in other words, s yields no information regarding which of its modular square roots was selected in Step 2). Thus, for every such s, the probability that either $\texttt{sqrt}(s, N)$ or $N - \texttt{sqrt}(s, N)$ equal the root chosen in Step 2 is at most 2/4. It follows that, on input a composite number, the reduction rejects with probability at least 1/2. ■

Reflection. Construction 4 illustrates an interesting aspect of randomized algorithms (or rather reductions); that is, the ability to hide information from a subroutine. Specifically, Construction 4 generates a problem instance (N, s) without disclosing any additional information. Furthermore, a correct solution to this instance is likely to help the reduction; that is, a correct answer to the instance (N, s) provides probabilistic evidence regarding whether N is a prime, where the probability space refers to the missing information (regarding how s was generated).

[4] This can be checked by scanning all possible powers $e \in \{2, ..., \log_2 N\}$, and (approximately) solving the equation $x^e = N$ for each value of e (i.e., finding the smallest integer i such that $i^e \geq N$). Such a solution can be found by binary search.

Comment. Testing primality is actually in $\mathcal{P}$, however, the deterministic algorithm demonstrating this fact is more complex (and its analysis is even more complicated).

3 One-sided error: The complexity classes RP and coRP

In this section we consider notions of probabilistic polynomial-time algorithms having one-sided error. The notion of one-sided error refers to a natural partition of the set of instances; that is, yes-instances versus no-instances in the case of decision problems, and instances having solution versus instances having no solution in the case of search problems. We focus on decision problems, and comment that an analogous treatment can be provided for search problems (see the second paragraph of Section 2).

Definition 6 (the class $\mathcal{RP}$)[5]: *A decision problem S is in $\mathcal{RP}$ if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that $\mathsf{Pr}[A(x) = 1] \geq 1/2$ and for every $x \notin S$ it holds that $\mathsf{Pr}[A(x)=0] = 1$.*

The choice of the constant $1/2$ is immaterial, and any other constant greater than zero will do (and yields the very same class). Similarly, this constant can be replaced by $1 - \mu(|x|)$ for various negligible functions μ (while preserving the class). Both facts are special cases of the robustness of the class (see Exercise 16).

Observe that $\mathcal{RP} \subseteq \mathcal{NP}$ (see Exercise 22) and that $\mathcal{RP} \subseteq \mathcal{BPP}$ (by the aforementioned error-reduction). Defining $\mathrm{co}\mathcal{RP} = \{\{0,1\}^* \setminus S : S \in \mathcal{RP}\}$, note that $\mathrm{co}\mathcal{RP}$ corresponds to the opposite direction of one-sided error probability. That is, *a decision problem S is in $\mathrm{co}\mathcal{RP}$ if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that $\mathsf{Pr}[A(x)=1] = 1$ and for every $x \notin S$ it holds that $\mathsf{Pr}[A(x)=0] \geq 1/2$.*

Relating BPP to RP

A natural question regarding probabilistic polynomial-time algorithms refers to the relation between two-sided and one-sided error probability. For example, *is $\mathcal{BPP}$ contained in $\mathcal{RP}$*? Loosely speaking, we show that $\mathcal{BPP}$ is reducible to $\mathrm{co}\mathcal{RP}$ by *one-sided error* randomized Karp-reductions. Note that $\mathcal{BPP}$ is trivially reducible to $\mathrm{co}\mathcal{RP}$ by *two-sided error* randomized Karp-reductions whereas a deterministic reduction of $\mathcal{BPP}$ to $\mathrm{co}\mathcal{RP}$ would imply $\mathcal{BPP} = \mathrm{co}\mathcal{RP} = \mathcal{RP}$ (see Exercise 19). The actual statement refers to the promise problem versions of both classes, which are briefly defined next.

First, we refer the reader to the general discussion of promise problems in [6]. We say that the promise problem $\Pi = (S_{\rm yes}, S_{\rm no})$ is in (promise problem extension of) $\mathcal{BPP}$ *if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S_{\rm yes}$ it holds that $\mathsf{Pr}[A(x) = 1] \geq 2/3$ and for every $x \in S_{\rm no}$ it holds that $\mathsf{Pr}[A(x) = 0] \geq 2/3$.* Similarly, Π is in $\mathrm{co}\mathcal{RP}$ if for every $x \in S_{\rm yes}$ it holds that $\mathsf{Pr}[A(x) = 1] = 1$ and for every $x \in S_{\rm no}$ it holds that $\mathsf{Pr}[A(x) = 0] \geq 1/2$. Probabilistic reductions among promise problems are defined by adapting the standrad conventions; specifically, queries that violate the promise at the target of the reduction may be answered arbitrarily.

[5]The initials RP stands for Random Polynomial-time, which fails to convey the restricted type of error allowed in this class. The only nice feature of this notation is that it is reminiscent of NP, thus reflecting the fact that $\mathcal{RP}$ is a randomized polynomial-time class contained in $\mathcal{RP}$.

Theorem 7 *Any problem in $\mathcal{BPP}$ is reducible by a one-sided error randomized Karp-reduction to* co$\mathcal{RP}$*, where* co$\mathcal{RP}$ *(and possibly also $\mathcal{BPP}$) denotes the corresponding class of promise problems. Specifically, the reduction always maps a* no*-instance to a* no*-instance.*

It follows that $\mathcal{BPP}$ is reducible by a one-sided error randomized Cook-reduction to $\mathcal{RP}$. Thus, referring to classes of promise problems, we may write $\mathcal{BPP} \subseteq \mathcal{RP}^{\mathcal{RP}}$. In fact, since $\mathcal{RP}^{\mathcal{RP}} \subseteq \mathcal{BPP}^{\mathcal{BPP}} = \mathcal{BPP}$, we have $\mathcal{BPP} = \mathcal{RP}^{\mathcal{RP}}$. Theorem 7 may be paraphrased as saying that the combination of the one-sided error probability of the reduction and the one-sided error probability of co$\mathcal{RP}$ can account for the two-sided error probability of $\mathcal{BPP}$. We warn that this statement is not a triviality like $1+1=2$, and in particular we do not know whether it holds for classes of standard decision problems (rather than for the classes of promise problems considered in Theorem 7).

Proof: Recall that we can easily reduce the error probability of BPP-algorithms, and derive probabilistic polynomial-time algorithms of exponentially vanishing error probability. But this does not eliminate the error (even on "one side") altogether. In general, there seems to be no hope to eliminate the error, unless we (either do something earth-shaking or) change the setting as done when allowing a one-sided error randomized reduction to a problem in co$\mathcal{RP}$. The latter setting can be viewed as a two-move randomized game (i.e., a random move by the reduction followed by a random move by the decision procedure of co$\mathcal{RP}$), and it enables applying different quantifiers to the two moves (i.e., allowing error in one direction in the first quantifier and error in the other direction in the second quantifier). In the next paragraph, which is inessential to the actual proof, we illustrate the potential power of this setting.

> **Teaching note:** The following illustration represents an alternative way of proving Theorem 7. This way seems conceptual simpler but it requires a starting point (or rather an assumption) that is much harder to establish, where both comparisons are with respect to the actual proof of Theorem 7 (which follows the illustration).

An illustration. Suppose that for some set $S \in \mathcal{BPP}$ there exists a polynomial p' and an off-line BPP-algorithm A' such that for every x it holds that $\mathsf{Pr}_{r\in\{0,1\}^{2p'(|x|)}}[A'(x,r)\neq\chi_S(x)] < 2^{-(p'(|x|)+1)}$; that is, the algorithm uses $2p'(|x|)$ bits of randomness and has error probability smaller than $2^{-p'(|x|)}/2$. Note that such an algorithm cannot be obtained by standard error-reduction (see Exercise 20). Anyhow (by applying a union bound), such a small error probability allows for meaningful outcomes even if only half of the string r is random. Thus, for every $x \in S$, it holds that $\mathsf{Pr}_{r'\in\{0,1\}^{p'(|x|)}}[(\forall r'' \in \{0,1\}^{p'(|x|)})\, A'(x,r'r'') = 1] > 1/2$, whereas for every $x \notin S$ and every $r' \in \{0,1\}^{p'(|x|)}$ it holds that $\mathsf{Pr}_{r''\in\{0,1\}^{p'(|x|)}}[A'(x,r'r'')=1] < 1/2$. That is, the error on yes-instances is "pushed" to the selection of r', whereas the error on no-instances is pushed to the selection of r''. This yields a one-sided error randomized Karp-reduction that maps x to (x,r'), where r' is uniformly selected in $\{0,1\}^{p'(|x|)}$, such that deciding S is reduced to the coRP problem (regarding pairs (x,r')) that is decided by the (on-line) randomized algorithm A'' defined by $A''(x,r') \stackrel{\rm def}{=} A'(x,r'U_{p'(|x|)})$. For details, see Exercise 21. The actual proof, which avoids the aforementioned hypothesis, follows.

The actual starting point. Consider any BPP-problem with a characteristic function χ (which, in case of a promise problem, is a partial function, defined only over the promise). By standard error-reduction, there exists a probabilistic polynomial-time algorithm A such that for every x on which χ is defined it holds that $\mathsf{Pr}[A(x) \neq \chi(x)] < \mu(|x|)$, where μ is a negligible function. Looking at the corresponding off-line algorithm A' and denoting by p the polynomial that bounds the running time of A, we have

$$\mathsf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A'(x,r)\neq\chi(x)] \;<\; \mu(|x|) \;<\; \frac{1}{2p(|x|)} \tag{1}$$

for all sufficiently long x's on which χ is defined. We show a randomized one-sided error Karp-reduction of χ to a promise problem in co$\mathcal{RP}$.

The main idea. As in the illustrating paragraph, the basic idea is "pushing" the error probability on yes-instances (of χ) to the reduction, while pushing the error probability on no-instances to the coRP-problem. Focusing on the case that $\chi(x) = 1$, this is achieved by augmenting the input x with a random sequence of "modifiers" that act on the random-input of algorithm A such that for a good choice of modifiers it holds that for every $r \in \{0,1\}^{p(|x|)}$ there exists a modifier in this sequence that when applied to r yields r' that satisfies $A(x, r') = 1$. Indeed, not all sequences of modifiers are good, but a random sequence will be good with high probability and bad sequences will be accounted for in the error probability of the reduction. On the other hand, using only modifiers that are permutations guarantees that the error probability on no-instances only increase by a factor that equals the number of modifiers we use, and this error probability will be accounted for by the error probability of the coRP-problem. Details follow.

The aforementioned modifiers are implemented by shifts (of the set of all strings by fixed offsets). Thus, we augment the input x with a random sequence of shifts, denoted $s_1, ..., s_m \in \{0,1\}^{p(|x|)}$, such that for a good choice of $(s_1, ..., s_m)$ it holds that for every $r \in \{0,1\}^{p(|x|)}$ there exists an $i \in [m]$ such that $A'(x, r \oplus s_i) = 1$. We will show that, for any yes-instance x and a suitable choice of m, with very high probability, a random sequence of shifts is good. Thus, for $A''(\langle x, s_1, ..., s_m \rangle, r) \stackrel{\text{def}}{=} \vee_{i=1}^{m} A'(x, r \oplus s_i)$, it holds that, with very high probability over the choice of $s_1, ..., s_m$, a yes-instance x is mapped to an augmented input $\langle x, s_1, ..., s_m \rangle$ that is accepted by A'' with probability 1. On the other hand, the acceptance probability of augmented no-instances (for any choice of shifts) only increases by a factor of m. In further detailing the foregoing idea, we start by explicitly stating the simple randomized mapping (to be used as a randomized Karp-reduction), and next define the target promise problem.

The randomized mapping. On input $x \in \{0,1\}^n$, we set $m = p(|x|)$, uniformly select $s_1, ..., s_m \in \{0,1\}^m$, and output the pair $(x, \overline{s})$, where $\overline{s} = (s_1, ..., s_m)$. Note that this mapping, denoted M, is easily computable by a probabilistic polynomial-time algorithm.

The promise problem. We define the following promise problem, denoted $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$, having instances of the form $(x, \overline{s})$ such that $|\overline{s}| = p(|x|)^2$.

- The yes-instances are pairs $(x, \overline{s})$, where $\overline{s} = (s_1, ..., s_m)$ and $m = p(|x|)$, such that for every $r \in \{0,1\}^m$ there exists an i satisfying $A'(x, r \oplus s_i) = 1$.
- The no-instances are pairs $(x, \overline{s})$, where again $\overline{s} = (s_1, ..., s_m)$ and $m = p(|x|)$, such that for at least half of the possible $r \in \{0,1\}^m$, for every i it holds that $A'(x, r \oplus s_i) = 0$.

To see that Π is indeed a co$\mathcal{RP}$ promise problem, we consider the following randomized algorithm. On input $(x, (s_1, ..., s_m))$, where $m = p(|x|) = |s_1| = \cdots = |s_m|$, the algorithm uniformly selects $r \in \{0,1\}^m$, and accepts if and only if $A'(x, r \oplus s_i) = 1$ for some $i \in \{1, ..., m\}$. Indeed, yes-instances of Π are accepted with probability 1, whereas no-instances of Π are rejected with probability at least 1/2.

Analyzing the reduction: We claim that the randomized mapping M reduces χ to Π with one-sided error. Specifically, we will prove two claims.

Claim 1: If x is a yes-instance (i.e., $\chi(x) = 1$) then $\Pr[M(x) \in \Pi_{\text{yes}}] > 1/2$.

Claim 2: If x is a no-instance (i.e., $\chi(x) = 0$) then $\Pr[M(x) \in \Pi_{\text{no}}] = 1$.

We start with Claim 2, which is easier to establish. Recall that $M(x) = (x, (s_1, ..., s_m))$, where $s_1, ..., s_m$ are uniformly and independently distributed in $\{0,1\}^m$. We note that (by Eq. (1) and $\chi(x) = 0$), for every possible choice of $s_1, ..., s_m \in \{0,1\}^m$ and every $i \in \{1, ..., m\}$, the fraction of r's that satisfy $A'(x, r \oplus s_i) = 1$ is at most $\frac{1}{2m}$. Thus, for every possible choice of $s_1, ..., s_m \in \{0,1\}^m$, the fraction of r's for which there exists an i such that $A'(x, r \oplus s_i) = 1$ holds is at most $1/2$. Hence, the reduction M always maps the no-instance x (i.e., $\chi(x) = 0$) to a no-instance of Π (i.e., an element of Π_{no}).

Turning to Claim 1 (which refers to $\chi(x) = 1$), we will show shortly that in this case, with very high probability, the reduction M maps x to a yes-instance of Π. We upper-bound the probability that the reduction fails (in case $\chi(x) = 1$) as follows:

$$\begin{aligned}
\mathsf{Pr}[M(x) \notin \Pi_{\text{yes}}] &= \mathsf{Pr}_{s_1,\ldots,s_m}[\exists r \in \{0,1\}^m \text{ s.t. } (\forall i)\ A'(x, r \oplus s_i) = 0] \\
&\leq \sum_{r \in \{0,1\}^m} \mathsf{Pr}_{s_1,\ldots,s_m}[(\forall i)\ A'(x, r \oplus s_i) = 0] \\
&= \sum_{r \in \{0,1\}^m} \prod_{i=1}^{m} \mathsf{Pr}_{s_i}[A'(x, r \oplus s_i) = 0] \\
&< 2^m \cdot \left(\frac{1}{2m}\right)^m
\end{aligned}$$

where the last inequality is due to Eq. (1). It follows that if $\chi(x) = 1$ then $\mathsf{Pr}[M(x) \in \Pi_{\text{yes}}] \gg 1/2$. Thus, the randomized mapping M reduces χ to Π, with one-sided error on yes-instances. Recalling that $\Pi \in \text{co}\mathcal{RP}$, the theorem follows. ■

Corollaries. The traditional presentation uses the foregoing reduction for showing that $\mathcal{BPP}$ is in the Polynomial-Time Hierarchy (where both classes refer to standard decision problems). Specifically, to prove that $\mathcal{BPP} \subseteq \Sigma_2$, define the polynomial-time computable predicate $\varphi(x, \overline{s}, r) \stackrel{\text{def}}{=} \bigvee_{i=1}^{m}(A'(x, s_i \oplus r) = 1)$, and observe that

$$\chi(x) = 1 \quad \Rightarrow \quad \exists \overline{s}\, \forall r \ \ \varphi(x, \overline{s}, r) \tag{2}$$

$$\chi(x) = 0 \quad \Rightarrow \quad \forall \overline{s}\, \exists r \ \ \neg\varphi(x, \overline{s}, r) \tag{3}$$

(where Eq. (3) is equivalent to $\neg\exists\overline{s}\,\forall r\ \ \varphi(x, \overline{s}, r)$). Note that Claim 1 (in the proof of Theorem 7) establishes that *most* sequences $\overline{s}$ satisfy $\forall r\, \varphi(x, \overline{s}, r)$, whereas Eq. (2) only requires the existence of *at least one* such $\overline{s}$. Similarly, Claim 2 establishes that for every $\overline{s}$ *most* choices of r violate $\varphi(x, \overline{s}, r)$, whereas Eq. (3) only requires that for every $\overline{s}$ there exists *at least one* such r. We comment that the same proof idea yields a variety of similar statements (e.g., $\mathcal{BPP} \subseteq \mathcal{MA}$, where $\mathcal{MA}$ is a randomized version of $\mathcal{NP}$).[6]

4 Zero-sided error: The complexity class ZPP

We now consider probabilistic polynomial-time algorithms that never err, but may fail to provide an answer. Focusing on decision problems, the corresponding class is denoted $\mathcal{ZPP}$ (standing for

[6]Specifically, the class $\mathcal{MA}$ is defined by allowing the verification algorithm V in the definition of $\mathcal{NP}$ to be probabilistic and err on no-instances; that is, for every $x \in S$ there exists $y \in \{0,1\}^{\text{poly}(|x|)}$ such that $\mathsf{Pr}[V(x,y) = 1] = 1$, whereas for every $x \notin S$ and every y it holds that $\mathsf{Pr}[V(x,y) = 0] \geq 1/2$. We note that $\mathcal{MA}$ can be viewed as a hybrid of the two aforementioned pairs of conditions; specifically, each problem in $\mathcal{MA}$ satisfy the conjunction of Eq. (2) and Claim 2. (Note that it makes no sense to require zero error on yes-instances, whereas the two-sided error version is equivalent to $\mathcal{MA}$; see Exercise 23.)

Zero-error Probabilistic Polynomial-time). The standard definition of $\mathcal{ZPP}$ is in terms of machines that output $\perp$ (indicating failure) with probability at most $1/2$. That is, *$S \in \mathcal{ZPP}$ if there exists a probabilistic polynomial-time algorithm A such that for every $x \in \{0,1\}^*$ it holds that $\Pr[A(x) \in \{\chi_S(x), \perp\}] = 1$ and $\Pr[A(x) = \chi_S(x)] \geq 1/2$, where $\chi_S(x) = 1$ if $x \in S$ and $\chi_S(x) = 0$ otherwise.* Again, the choice of the constant (i.e., $1/2$) is immaterial, and "error-reduction" can be performed showing that algorithms that yield a meaningful answer with noticeable probability can be amplified to algorithms that fail with negligible probability (see Exercise 17).

Theorem 8 *$\mathcal{ZPP} = \mathcal{RP} \cap \mathrm{co}\mathcal{RP}$.*

Proof Sketch: The fact that $\mathcal{ZPP} \subseteq \mathcal{RP}$ (as well as $\mathcal{ZPP} \subseteq \mathrm{co}\mathcal{RP}$) follows by a trivial transformation of the ZPP-algorithm; that is, replacing the failure indicator $\perp$ by a "no" verdict (resp., "yes" verdict). Note that the choice of what to say in case the ZPP-algorithm fails is determined by the type of error that we are allowed.

In order to prove that $\mathcal{RP} \cap \mathrm{co}\mathcal{RP} \subseteq \mathcal{ZPP}$ we combine the two algorithm guaranteed for a set in $\mathcal{RP} \cap \mathrm{co}\mathcal{RP}$. The point is that we can trust the RP-algorithm (resp., coNP-algorithm) in the case that it says "yes" (resp., "no"), but not in the case that it says "no" (resp., "yes"). Thus, we invoke both algorithms, and output a definite answer only if we obtain an answer that we can trust (which happen with high probability). Otherwise, we output $\perp$. $\square$

Expected polynomial-time. In some sources $\mathcal{ZPP}$ is defined in terms of randomized algorithms that run in expected polynomial-time and always output the correct answer. This definition is equivalent to the one we used (see Exercise 18).

5 Randomized Log-Space

In this section we discuss probabilistic polynomial-time algorithms that are further restricted such that they are allowed to use only a logarithmic amount of space.

5.1 Definitional issues

When defining space-bounded randomized algorithms, we face a problem analogous to the one discussed in the context of non-deterministic space-bounded computation. Specifically, the on-line and the off-line versions (formulated in Definition 1) are no longer equivalent, unless we restrict the off-line machine to access its random-input tape in a uni-directional manner. The issue is that, in the context of space-bounded computation (and unlike in the case that we only care about time-bounds), the results of the internal coin tosses (in the on-line model) cannot be recorded for free. Bearing in mind that, *in the current context*, we wish to model real algorithms (rather than present a fictitious model that captures a fundamental phenomena as in the case of space complexity), it is clear that *using the on-line version is the natural choice.*

An additional issue that arises is the need to explicitly bound the running-time of space-bounded randomized algorithms. Recall that, without loss of generality, the number of steps taken by a space-bounded non-deterministic machine is at most exponential in its space complexity, because the shortest path between two configurations in the (directed) graph of possible configurations is upper-bounded by its size (which in turn is exponential in the space-bound). This reasoning fails in the case of randomized algorithms, because the shortest path between two configurations does not bound the expected number of random steps required for going from the first configuration to the second one. In fact, as we shall shortly see, failing to upper-bound the running time of log-space randomized algorithms seems to allow them too much power; that is, such (unrestricted) log-space randomized algorithms can emulate non-deterministic log-space computations (in exponential time). The emulation consists of repeatedly invoking the NL-machine, while using random choices in the role of the non-deterministic moves. If the input is a yes-instance then, in each attempt, with probability at least 2^{-t}, we "hit" an accepting t-step (non-deterministic) computation, where t is polynomial in the input length. Thus, the randomized machine accepts such a yes-instance after an expected number of 2^t trials. To allow for the rejection of no-instances (rather than looping infinitely in vain), we wish to implement a counter that counts till 2^t (or so) and reject the input if this number of trials have failed. We need to implement such a counter within space $O(\log t)$ rather than t (which is easy). In fact, it suffices to have a "randomized counter" that, with high probability, counts to approximately 2^t. The implementation of such a counter is left to Exercise 26, and using it we may obtain a randomized algorithm that halts with high probability (on every input), always rejects a no-instance, and accepts each yes-instance with probability at least 1/2.

In light of the foregoing discussion, when defining randomized log-space algorithms we explicitly require that the algorithms halt in polynomial-time. Under this convention, the class $\mathcal{RL}$ relates to $\mathcal{NL}$ analogously to the relation of $\mathcal{RP}$ to $\mathcal{NP}$. That is, the probabilistic acceptance condition in $\mathcal{RL}$ is as in the case of $\mathcal{RP}$. The class $\mathcal{BPL}$ is defined similarly, when using a probabilistic acceptance condition as in the case of $\mathcal{BPP}$.

Definition 9 (the classes $\mathcal{RL}$ and $\mathcal{BPL}$): *We say that a randomized log-space algorithm is* admissible *if it always halts in a polynomial number of steps.*

- *A decision problem S is in $\mathcal{RL}$ if there exists an admissible* (on-line) *randomized log-space algorithm A such that for every $x \in S$ it holds that* $\Pr[A(x) = 1] \geq 1/2$ *and for every $x \notin S$ it holds that* $\Pr[A(x) = 0] = 1$.
- *A decision problem S is in $\mathcal{BPL}$ if there exists an admissible* (on-line) *randomized log-space algorithm A such that for every $x \in S$ it holds that* $\Pr[A(x) = 1] \geq 2/3$ *and for every $x \notin S$ it holds that* $\Pr[A(x) = 0] \geq 2/3$.

Clearly, $\mathcal{RL} \subseteq \mathcal{NL} \subseteq \mathcal{P}$ and $\mathcal{BPL} \subseteq \mathcal{P}$. Note that the classes $\mathcal{RL}$ and $\mathcal{BPL}$ remain unchanged even if we allow the algorithms to run for *expected* polynomial-time and have non-halting computations. Such algorithms can be easily transformed into admissible algorithms by truncating long computations, while using a (standard) counter (which can be implemented in logarithmic-space). Also note that error-reduction is applicable in the current setting (while essentially preserving both the time and space bounds).

5.2 The accidental tourist sees it all

An appealing example of a randomized log-space algorithm is presented next. It refers to the problem of deciding undirected connectivity, and demonstrated that this problem is in $\mathcal{RL}$. (Recall that this problem is actually in $\mathcal{L}$, but the algorithm and its analysis were more complicated.) Recall that Directed Connectivity is complete for $\mathcal{NL}$ (under log-space reductions). For sake of simplicity, we consider the following version of undirected connectivity, which is equivalent under log-space reductions to the version in which one needs to determine whether or not the input (undirected) graph is connected. In the current version, *the input consists of a triple (G, s, t), where G is an undirected graph, s, t are two vertices in G, and the task is to determine whether or not s and t are connected in G.*

Construction 10 *On input (G,s,t), the randomized algorithm starts a* poly$(|G|)$*-long random walk at vertex s, and accepts the triplet if and only if the walk passed through the vertex t. By a random walk we mean that at each step the algorithm selects uniformly one of the neighbors of the current vertex and moves to it.*

Observe that the algorithm can be implemented in logarithmic space (because we only need to store the current vertex as well as the number of steps taken so far). Obviously, if s and t are not connected in G then the algorithm always rejects (G,s,t). Proposition 11 implies that undirected connectivity is indeed in $\mathcal{RL}$.

Proposition 11 *If s and t are connected in $G=(V,E)$ then a random walk of length $O(|V|\cdot|E|)$ starting at s passes through t with probability at least $1/2$.*

In other words, a random walk starting at s visits all vertices of the connected component of s (i.e., it sees all that there is to see).

Proof Sketch: We will actually show that if G is connected then, with probability at least $1/2$, a random walk starting at s visits all the vertices of G. For any pair of vertices (u,v), let $X_{u,v}$ be a random variable representing the number of steps taken in a random walk starting at u until v is *first encountered*. The reader may verify that for every edge $\{u,v\}\in E$ it holds that $\mathrm{E}[X_{u,v}]\leq 2|E|$; see Exercise 27. Next, we let cover(G) denote the expected number of steps in a random walk starting at s and ending when the last of the vertices of V is encountered. Our goal is to upper-bound cover(G). Towards this end, we consider an arbitrary directed cyclic-tour C that visits all vertices in G, and note that

$$\mathrm{cover}(G) \;\leq\; \sum_{(u,v)\in C}\mathrm{E}[X_{u,v}] \;\leq\; |C|\cdot 2|E|.$$

In particular, selecting C as a traversal of some spanning tree of G, we conclude that cover$(G) < 4\cdot|V|\cdot|E|$. Thus, with probability at least $1/2$, a random walk of length $8\cdot|V|\cdot|E|$ starting at s visits all vertices of G. □

Notes

Making people take an unconventional step requires compelling reasons, and indeed the study of randomized algorithms was motivated by a few compelling examples. Ironically, the appeal of the two most famous examples (discussed next) has diminished due to subsequent finding, but the fundamental questions that emerged remain fascinating regardless of the status of these and other appealing examples (see Section 2.1).

The first example: primality testing. For more than two decades, primality testing was the archetypical example of the usefulness of randomization in the context of efficient algorithms. The celebrated algorithms of Solovay and Strassen [15] and of Rabin [11], proposed in the late 1970's, established that deciding primality is in co$\mathcal{RP}$ (i.e., these tests always recognize correctly prime numbers, but they may err on composite inputs). (The approach of Construction 4, which only establishes that deciding primality is in $\mathcal{BPP}$, is commonly attributed to M. Blum.) In the late 1980's, Adleman and Huang [1] proved that deciding primality is in $\mathcal{RP}$ (and thus in $\mathcal{ZPP}$). In the early 2000's, Agrawal, Kayal, and Saxena [2] showed that deciding primality is actually in $\mathcal{P}$. One should note, however, that strong evidence to the fact that deciding primality is in $\mathcal{P}$ was actually available from the start: we refer to Miller's deterministic algorithm [9], which relies on the Extended Riemann Hypothesis.

The second example: undirected connectivity. Another celebrated example to the power of randomization, specifically in the context of log-space computations, was provided by testing undirected connectivity. The random-walk algorithm presented in Construction 10 is due to Aleliunas, Karp, Lipton, Lovász, and Rackoff [3]. Recall that a deterministic log-space algorithm was found twenty-five years later (see [12]).

Other randomized algorithms. Although randomized algorithms are more abundant in the context of approximation problems (let alone in other computational settings (cf. Section 2.1)), quite a few such algorithms are known also in the context of search and decision problems. We mention the algorithms for finding perfect matchings and minimum cuts in graphs (see, e.g., [5, Apdx. B.1] or [10, Sec. 12.4&10.2]), and note the prominent role of randomization in computational number theory (see, e.g., [4] or [10, Chap. 14]). For a general textbook on randomized algorithms, we refer the interested reader to [10].

On the general study of $\mathcal{BPP}$. Turning to the general study of $\mathcal{BPP}$, we note that our presentation of Theorem 7 follows the proof idea of Lautemann [8]. A different proof technique, which yields a weaker result but found more applications (e.g., in the context of approximate counting and interactive proof systems), was presented (independently) by Sipser [14].

On the role of promise problems. In addition to their use in the formulation of Theorem 7, promise problems allow for establishing time hierarchy theorems for randomized computation. We mention that such results are not known for the corresponding classes of standard decision problems. The technical difficulty is that we do not know how to enumerate probabilistic machines that utilize a non-trivial probabilistic decision rule.

On the feasibility of randomized computation. Different perspectives on this question are offered by [5, Chap. 3]. Specifically, generating uniformly distributed bit sequences is not really necessary for implementing randomized algorithms; it suffices to generate sequences that look as if they are uniformly distributed. A less radical approach (see [13]) deals with the task of extracting almost uniformly distributed bit sequences from sources of weak randomness. Needless to say, these two approaches are complimentary and can be combined.

Exercises

Exercise 12 Show that if a search (resp., decision) problem can be solved by a probabilistic polynomial-time algorithm having zero error probability, then the problem can be solve by a deterministic polynomial-time algorithm.
(Hint: replace the internal coin tosses by a fixed outcome.)

Exercise 13 (randomized reductions) In continuation to the definitions presented at the beginning of the main text, prove the following:

1. If a problem Π is probabilistic polynomial-time reducible to a problem that is solvable in probabilistic polynomial-time then Π is solvable in probabilistic polynomial-time, where by solving we mean solving correctly except with negligible probability.

Warning: Recall that in the case that Π' is a search problem, we required that on input x the solver provides a correct solution with probability at least $1 - \mu(|x|)$, but we did not require that it always returns the same solution.

(Hint: without loss of generality, the reduction does not make the same query twice.)

2. Prove that probabilistic polynomial-time reductions are transitive.

3. Prove that randomized Karp-reductions are transitive and that they yield a special case of probabilistic polynomial-time reductions.

Define one-sided error and zero-sided error randomized (Karp and Cook) reductions, and consider the foregoing items when applied to them. Note that the implications for the case of one-sided error are somewhat subtle.

Exercise 14 (on the definition of probabilistically solving a search problem) In continuation to the discussion at the beginning of Section 2, suppose that for some probabilistic polynomial-time algorithm A and a positive polynomial p the following holds: for every $x \in S_R \stackrel{\text{def}}{=} \{z : R(z) \neq \emptyset\}$ there exists $y \in R(x)$ such that $\Pr[A(x) = y] > 0.5 + (1/p(|x|))$, whereas for every $x \notin S_R$ it holds that $\Pr[A(x) = \perp] > 0.5 + (1/p(|x|))$.

1. Show that there exists a probabilistic polynomial-time algorithm that solves the search problem of R with negligible error probability.

   (Hint: See Exercise 15 for a related procedure.)

2. Reflect on the need to require that one (correct) solution occurs with probability greater than $0.5 + (1/p(|x|))$. Specifically, what can we do if it is only guaranteed that for every $x \in S_R$ it holds that $\Pr[A(x) \in R(x)] > 0.5 + (1/p(|x|))$ (and for every $x \notin S_R$ it holds that $\Pr[A(x) = \perp] > 0.5 + (1/p(|x|))$)?

Exercise 15 (error-reduction for $\mathcal{BPP}$) For $\varepsilon : \mathbb{N} \to [0,1]$, let $\mathcal{BPP}_\varepsilon$ denote the class of decision problems that can be solved in probabilistic polynomial-time with error probability upper-bounded by ε. Prove the following two claims:

1. For every positive polynomial p and $\varepsilon(n) = (1/2) - (1/p(n))$, the class $\mathcal{BPP}_\varepsilon$ equals $\mathcal{BPP}$.

2. For every positive polynomial p and $\varepsilon(n) = 2^{-p(n)}$, the class $\mathcal{BPP}$ equals $\mathcal{BPP}_\varepsilon$.

Formulate a corresponding version for the setting of search problem. Specifically, for every input that has a solution, consider the probability that a specific solution is output.

Guideline: Given an algorithm A for the syntactically weaker class, consider an algorithm A' that on input x invokes A on x for $t(|x|)$ times, and rules by majority. For Part 1 set $t(n) = O(p(n)^2)$ and apply Chebyshev's Inequality. For Part 2 set $t(n) = O(p(n))$ and apply the Chernoff Bound.

Exercise 16 (error-reduction for $\mathcal{RP}$) For $\rho : \mathbb{N} \to [0,1]$, we define the class of decision problem $\mathcal{RP}_\rho$ such that it contains S if there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that $\Pr[A(x) = 1] \geq \rho(|x|)$ and for every $x \notin S$ it holds that $\Pr[A(x) = 0] = 1$. Prove the following two claims:

1. For every positive polynomial p, the class $\mathcal{RP}_{1/p}$ equals $\mathcal{RP}$.

2. For every positive polynomial p, the class $\mathcal{RP}$ equals $\mathcal{RP}_\rho$, where $\rho(n) = 1 - 2^{-p(n)}$.

(Hint: The one-sided error allows using an "or-rule" (rather than a "majority-rule") for the decision.)

Exercise 17 (error-reduction for $\mathcal{ZPP}$) For $\rho : \mathbb{N} \to [0,1]$, we define the class of decision problem $\mathcal{ZPP}_\rho$ such that it contains S if there exists a probabilistic polynomial-time algorithm A such that for every x it holds that $\Pr[A(x) = \chi_S(x)] \geq \rho(|x|)$ and $\Pr[A(x) \in \{\chi_S(x), \perp\}] = 1$, where $\chi_S(x) = 1$ if $x \in S$ and $\chi_S(x) = 0$ otherwise. Prove the following two claims:

1. For every positive polynomial p, the class $\mathcal{ZPP}_{1/p}$ equals $\mathcal{ZPP}$.
2. For every positive polynomial p, the class $\mathcal{ZPP}$ equals $\mathcal{ZPP}_\rho$, where $\rho(n) = 1 - 2^{-p(n)}$.

Exercise 18 (an alternative definition of $\mathcal{ZPP}$) We say that the decision problem S is solvable in expected probabilistic polynomial-time if there exists a randomized algorithm A and a polynomial p such that for every $x \in \{0,1\}^*$ it holds that $\Pr[A(x) = \chi_S(x)] = 1$ and the expected number of steps taken by $A(x)$ is at most $p(|x|)$. Prove that $S \in \mathcal{ZPP}$ if and only if S is solvable in expected probabilistic polynomial-time.

Guideline: Repeatedly invoking a ZPP algorithm until it yields an output other than $\perp$, results in an expected probabilistic polynomial-time solver. On the other hand, truncating runs of an expected probabilistic polynomial-time algorithm once they exceed twice the expected number of steps (and outputting $\perp$ on such runs), we obtain a ZPP algorithm.

Exercise 19 Let $\mathcal{BPP}$ and co$\mathcal{RP}$ be classes of promise problems (as in Theorem 7).

1. Prove that every problem in $\mathcal{BPP}$ is reducible to the set $\{1\} \in \mathcal{P}$ by a *two-sided error* randomized Karp-reduction.

   (Hint: Such a reduction may effectively decide membership in any set in $\mathcal{BPP}$.)

2. Prove that if a set S is Karp-reducible to $\mathcal{RP}$ (resp., co$\mathcal{RP}$) via a deterministic reduction then $S \in \mathcal{RP}$ (resp., $S \in$ co$\mathcal{RP}$).

Exercise 20 (randomness-efficient error-reductions) Note that standard error-reduction (see Exercise 15) yields error probability δ at the cost of increasing the randomness complexity by a *factor* of $O(\log(1/\delta))$. Using a randomness-efficient error-reductions, show that error probability δ can be obtained at the cost of increasing the randomness complexity by a constant factor and an *additive term* of $1.5\log_2(1/\delta)$. Note that this allows satisfying the hypothesis made in the illustrative paragraph of the proof of Theorem 7.

Exercise 21 In continuation to the illustrative paragraph of the proof of Theorem 7, consider the promise problem $\Pi' = (\Pi'_{\rm yes}, \Pi'_{\rm no})$ such that $\Pi'_{\rm yes} = \{(x,r') : |r'| = p(|x|)/2 \wedge (\forall r'' \in \{0,1\}^{|r'|})\, A'(x, r'r'') = 1\}$ and $\Pi'_{\rm no} = \{(x,r') : x \notin S\}$. Recall that for every x it holds that $\Pr_{r \in \{0,1\}^{p(|x|)}}[A'(x,r) \neq \chi_S(x)] < 2^{-((p(|x|)/2)+1)}$.

1. Show that mapping x to (x,r'), where r' is uniformly distributed in $\{0,1\}^{p(|x|)/2}$, constitutes a one-sided error randomized Karp-reduction of S to Π'.

2. Show that Π' is in the promise problem class co$\mathcal{RP}$.

Exercise 22 Prove that for every $S \in \mathcal{NP}$ there exists a probabilistic polynomial-time algorithm A such that for every $x \in S$ it holds that $\Pr[A(x) = 1] > 0$ and for every $x \notin S$ it holds that $\Pr[A(x) = 0] = 1$. That is, A has error probability at most $1 - \exp(-\text{poly}(|x|))$ on yes-instances but never errs on no-instances. Thus, $\mathcal{NP}$ may be fictitiously viewed as having a huge one-sided error probability.

Exercise 23 (randomized versions of $\mathcal{NP}$) In continuation to Footnote 6, consider the following two variants of $\mathcal{MA}$ (which we consider the main randomized version of $\mathcal{NP}$).

1. $S \in \mathcal{MA}^{(1)}$ if there exists a probabilistic polynomial-time algorithm V such that for every $x \in S$ there exists $y \in \{0,1\}^{\text{poly}(|x|)}$ such that $\Pr[V(x,y)=1] \geq 1/2$, whereas for every $x \notin S$ and every y it holds that $\Pr[V(x,y)=0] = 1$.

2. $S \in \mathcal{MA}^{(2)}$ if there exists a probabilistic polynomial-time algorithm V such that for every $x \in S$ there exists $y \in \{0,1\}^{\text{poly}(|x|)}$ such that $\Pr[V(x,y)=1] \geq 2/3$, whereas for every $x \notin S$ and every y it holds that $\Pr[V(x,y)=0] \geq 2/3$.

Prove that $\mathcal{MA}^{(1)} = \mathcal{NP}$ whereas $\mathcal{MA}^{(2)} = \mathcal{MA}$.

Guideline: For the first part, note that a sequence of internal coin tosses that makes V accept (x,y) can be incorporated into y itself (yielding a standard NP-witness). For the second part, apply the ideas underlying the proof of Theorem 7.

Exercise 24 (time hierarchy theorems for promise problem versions of BPTIME) Fixing a model of computation, let BPTIME(t) denote the class of promise problems that are solvable by a randomized algorithm of time complexity t that has a two-sided error probability at most 1/3. (The common definition refers only to decision problems.) Formulate and prove results analogous to those stated and proved for deterministic algorithms.

Guideline: Analogously to the proof of the deterministic time hierarchy, we construct a Boolean function f by associating with each admissible machine M an input x_M, and making sure that $\Pr[f(x_M) \neq M'(x)] \geq 2/3$, where $M'(x)$ denotes the emulation of $M(x)$ suspended after $t_1(|x|)$ steps. The key point is that f is a partial function (corresponding to a promise problem) that is defined only for machines that have two-sided error at most 1/3 (on every input). This allows for a randomized computation of f with two-sided error probability at most 1/3 (on each input on which f is defined).

Exercise 25 (extracting square roots modulo a prime) Using the following guidelines, present a probabilistic polynomial-time algorithm that, on input a prime P and a quadratic residue $s \pmod P$, returns r such that $r^2 \equiv s \pmod P$.

1. Prove that if $P \equiv 3 \pmod 4$ then $s^{(P+1)/4} \bmod P$ is a square root of the quadratic residue $s \pmod P$.

2. Note that the procedure suggested in Item 1 relies on the ability to find an odd power e such that $s^e \equiv 1 \pmod P$. (In Item 1, we used $e = (P-1)/2$, which is odd since $P \equiv 3 \pmod 4$.) Once such a power is found, we may output $s^{(e+1)/2} \bmod P$.

   Show that it suffices to find an odd power e together with a residue r and an even power e' such that $s^e r^{e'} \equiv 1 \pmod P$.

3. Given a prime $P \equiv 1 \pmod 4$, a quadratic residue s, and a quadratic non-residue r (equiv., $r^{(P-1)/2} \equiv -1 \pmod P$), show that e and e' as in Item 2 can be efficiently found.[7]

4. Prove that, for a prime P, with probability $1/2$ a uniformly chosen $r \in \{1, ..., P\}$ satisfies $r^{(P-1)/2} \equiv -1 \pmod P$.

Note that randomization is used only in the last item, which in turn is used only for $P \equiv 1 \pmod 4$.

Exercise 26 (small-space randomized step-counter) A step-counter is an algorithm that runs for a number of steps that is specified in its input. Actually, such an algorithm may run for a somewhat larger number of steps but halt after issuing a number of "signals" as specified in its input, where these signals are defined as entering (and leaving) a designated state (of the algorithm). A step-counter may be run in parallel to another procedure in order to suspend the execution after a desired number of steps (of the other procedure) has elapsed. We note that there exists a simple deterministic machine that, on input n, halts after issuing n signals while using $O(1) + \log_2 n$ space (and $\widetilde{O}(n)$ time). The goal of this exercise is presenting a (randomized) step-counter that allows for many more signals while using the same amount of space. Specifically, present a (randomized) algorithm that, on input n, uses $O(1) + \log_2 n$ space (and $\widetilde{O}(2^n)$ time) and halts after issuing an expected number of 2^n signals. Furthermore, prove that, with probability at least $1 - 2^{-k+1}$, this step-counter halts after issuing a number of signals that is between 2^{n-k} and 2^{n+k}.

Guideline: Repeat the following experiment till reaching success. Each trial consists of uniformly selecting n bits (i.e., tossing n unbiased coins), and is deemed successful if all bits turn out to equal the value 1 (i.e., all outcomes equal HEAD). Note that such a trial can be implemented by using space $O(1) + \log_2 n$ (mainly for implementing a standard counter for determining the number of bits). Thus, each trial is successful with probability 2^{-n}, and the expected number of trials is 2^n.

Exercise 27 (analysis of random walks on arbitrary undirected graphs) In order to complete the proof of Proposition 11, prove that if $\{u, v\}$ is an edge of the graph $G = (V, E)$ then $\mathrm{E}[X_{u,v}] \leq 2|E|$. Recall that, for a fixed graph, $X_{u,v}$ is a random variable representing the number of steps taken in a random walk that starts at the vertex u until the vertex v is first encountered.

Guideline: Let $Z_{u,v}(n)$ be a random variable counting the number of *minimal* paths from u to v that appear along a random walk of length n, where the walk starts at the stationary vertex distribution (which is well-defined assuming the graph is not bipartite, which in turn may be enforced by adding a self-loop). On one hand, $\mathsf{E}[X_{u,v} + X_{v,u}] = \lim_{n\to\infty}(n/\mathsf{E}[Z_{u,v}(n)])$, due to the memoryless property of the walk. On the other hand, letting $\chi_{v,u}(i) \stackrel{\text{def}}{=} 1$ if the edge $\{u, v\}$ was traversed from v to u in the i^{th} step of such a random walk and $\chi_{v,u}(i) \stackrel{\text{def}}{=} 0$ otherwise, we have $\sum_{i=1}^{n} \chi_{v,u}(i) \leq Z_{u,v}(n) + 1$ and $\mathrm{E}[\chi_{v,u}(i)] = 1/2|E|$ (because, in each step, each directed edge appears on the walk with equal probability). It follows that $\mathrm{E}[X_{u,v}] < 2|E|$.

Exercise 28 (the class $\mathcal{PP} \supseteq \mathcal{BPP}$ and its relation to $\#\mathcal{P}$) In contrast to $\mathcal{BPP}$, which refers to useful probabilistic polynomial-time algorithms, the class $\mathcal{PP}$ does not capture such algorithms but is rather closely related to $\#\mathcal{P}$. A decision problem S is in $\mathcal{PP}$ if there exists a probabilistic

[7]Write $(P-1)/2 = (2j+1) \cdot 2^{i_0}$, and note that $s^{(2j+1)\cdot 2^{i_0}} \equiv 1 \pmod P$. Assuming we have that if for some $i' > i > 0$ and j' it holds that $s^{(2j+1)\cdot 2^{i}} r^{(2j'+1)\cdot 2^{i'}} \equiv 1 \pmod P$, show how to find $i'' > i-1$ and j'' such that $s^{(2j+1)\cdot 2^{i-1}} r^{(2j''+1)\cdot 2^{i''}} \equiv 1 \pmod P$. (Extra hint: $s^{(2j+1)\cdot 2^{i-1}} r^{(2j'+1)\cdot 2^{i'-1}} \equiv \pm 1 \pmod P$ and $r^{(2j+1)\cdot 2^{i_0}} \equiv -1 \pmod P$.) Note that for $i = 1$ we obtain what we need.

polynomial-time algorithm A such that, for every x, it holds that $x \in S$ if and only if $\Pr[A(x) = 1] > 1/2$. Note that $\mathcal{BPP} \subseteq \mathcal{PP}$. Prove that $\mathcal{PP}$ is Cook-reducible to $\#\mathcal{P}$ and vise versa.

Guideline: For $S \in \mathcal{PP}$ (by virtue of the algorithm A), consider the relation R such that $(x, r) \in R$ if and only if A accepts the input x when using the random-input $r \in \{0,1\}^{p(|x|)}$, where p is a suitable polynomial. Thus, $x \in S$ if and only if $|R(x)| > 2^{p(|x|)-1}$, which in turn can de determined by querying the counting function of R. To reduce $f \in \#\mathcal{P}$ to $\mathcal{PP}$, consider the relation $R \in \mathcal{PC}$ that is counted by f (i.e., $f(x) = |R(x)|$) and the decision problem $S_f = \{(x, v) : f(x) \geq v\}$. Let p be the polynomial specifying the length of solutions for R (i.e., $(x, y) \in R$ implies $|y| = p(|x|)$), and consider the algorithm A' that on input (x, N) proceeds as follows: With probability $1/2$, it uniformly selects $y \in \{0,1\}^{p(|x|)}$ and accepts if and only if $(x, y) \in R$, and otherwise (i.e., in the other case) it accepts with probability $\frac{2^{p(|x|)}-N+0.5}{2^{p(|x|)}}$. Prove that $(x, N) \in S_f$ if and only if $\Pr[A'(x) = 1] > 1/2$.

References

[1] L.M. Adleman and M. Huang. *Primality Testing and Abelian Varieties Over Finite Fields.* Springer-Verlag Lecture Notes in Computer Science (Vol. 1512), 1992. Preliminary version in *19th STOC*, 1987.

[2] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, Vol. 160 (2), pages 781–793, 2004.

[3] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[4] E. Bach and J. Shallit. *Algorithmic Number Theory* (Volume I: Efficient Algorithms). MIT Press, 1996.

[5] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), Springer, 1999.

[6] O. Goldreich. On Promise Problems (a survey in memory of Shimon Even [1935-2004]). *ECCC*, TR05-018, 2005.

[7] S. Goldwasser and M. Sipser. Private Coins versus Public Coins in Interactive Proof Systems. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 73–90, 1989. Extended abstract in *18th STOC*, 1986.

[8] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, 17, pages 215–217, 1983.

[9] G.L. Miller. Riemann's Hypothesis and Tests for Primality. *Journal of Computer and System Science*, Vol. 13, pages 300–317, 1976.

[10] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[11] M.O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, Vol. 12, pages 128–138, 1980.

[12] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[13] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. *Bulletin of the EATCS 77*, pages 67–95, 2002.

[14] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[15] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, Vol. 6, pages 84–85, 1977. Addendum in *SIAM Journal on Computing*, Vol. 7, page 118, 1978.